TAMPEREEN TEKNILLINEN YLIOPISTO

TIMO POKKINEN

COBOL-OHJELMISTOJEN SIIRRETTÄVYYDESTÄ

Diplomityö

Tarkastajat: professori Tommi Mikkonen, professori Seppo Kuikka ja Visa-Matti Leinikki, Tietonauha-yhtiö
Tarkastajat ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 7. huhtikuuta 2010

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
POKKINEN, TIMO: COBOL-ohjelmistojen siirrettävyydestä
Diplomityö, 48 sivua, 9 liitesivua
Toukokuu 2010
Pääaine: Ohjelmistotekniikka
Tarkastajat: professori Tommi Mikkonen, professori Seppo Kuikka ja Visa-Matti Leinikki, Tietonauha-yhtiö
Avainsanat: COBOL, Siirrettävyys, HP3000

Ohjelmistojen siirrettävyys on erityisen tärkeää pitkän elinkaaren omaavien ohjelmistojen toteutuksessa. Siirrettävyys varmistaa ohjelmiston käyttökelpoisuuden laitteiden, käyttöjärjestelmien, varusohjelmistojen ja ohjelmistokehitysvälineiden muutosten yhteydessä. Ohjelmiston siirrettävyys tarkoittaa sitä, että ohjelmisto voidaan pienin muutoksin tai kokonaan ilman muutoksia siirtää toiseen tietokoneeseen ja käyttöympäristöön. Siirrettävyydestä käytetään englanninkielisessä kirjallisuudessa termejä portability, transferability ja movability. Ohjelmistot, joiden kohdalla siirrettävyysongelmista selviytyminen on erityisen tärkeää, ovat usein logiikaltaan monimutkaisia ja niiden toiminnallisuutta ohjaavat ja rajoittavat lait sekä asetukset.

Tämä diplomityö liittyy Tietonauha-yhtiö Oy:n COBOL-ohjelmointikielellä toteutetun palkka- ja henkilöstöhallinnan ohjelmiston kehitysprojektiin, jossa COBOL-ohjelmisto TIPA muokattiin suljetun ohjelmointiympäristön ohjelmistosta siirrettäväksi. Ohjelmia TIPAssa on yli 1200 ja COBOL-koodia yli 300 000 riviä. TIPAa on kehitetty yli 40 henkilötyövuoden työpanoksella, ja suuri osa työstä on mennyt järjestelmän lakisääteisten ominaisuuksien toteuttamiseen ja järjestelmän sovittamiseen useiden erilaisten työehtosopimusten ja toimintatapojen mukaiseksi.

Teknisellä tasolla diplomityössä tutkittiin siirrettävyyteen liittyviä ongelmia, ratkaistiin TIPA-järjestelmän siirrettävyyteen liittyvät ongelmat ja muutettiin suljetun järjestelmän COBOL-ohjelmisto siirrettäväksi. Siirrettävällä ohjelmistoversiolla saavutettiin yksi helposti ylläpidettävä ohjelmistoversio ja laaja tuettu laite- sekä käyttöjärjestelmäkanta. Lopputuloksena TIPA-ohjelmistolle tuli yli kymmenen vuotta lisää käyttöikää ja ohjelmiston potentiaalinen asiakaskunta laajeni moninkertaiseksi. Työssä tarkastellaan ohjelmointivälineiden antamaa tukea siirrettäville ohjelmistoille ja säilytettiin järjestelmien kehitetykseen käytetty useiden vuosien työ sekä etu yrityksen henkilöstön vahvasta COBOL-ohjelmointiosaamisesta.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
POKKINEN, TIMO: Study of Portability in COBOL applications
Master of Science Thesis, 48 pages, 9 Appendix pages
May 2010
Major: Software engineering
Examiners: Professor Tommi Mikkonen, Professor Seppo Kuikka and Visa-Matti Leinikki, Tietonauha-yhtiö Oy
Keywords: COBOL, Portability, Transferability, HP3000

Portability of software applications is particularly important with applications having a long lifecycle. Portability ensures usability of an application while hardware, operating systems, firmware and software development tools change. Portability means that a software application can be moved to another computer setting with minor changes or even without changes at all. Terms used with portability are transferability and movability. Software applications that are most affected by portability problems are applications with large amount of business logic and applications constrained by laws and regulations.

This thesis is a part of a software development project where Tietonauha-yhtiö's payroll and human resources application TIPA, developed with COBOL, was upgraded from a closed development platform to a portable application. There are over 1200 programs and over 300 000 lines of COBOL code in TIPA. TIPA has been developed over 40 man years and large part of that work has been used to produce a software product compliant with a vast amount of collective agreements and regulations.

In this thesis a study of portability problems was carried out. The problems with TIPA were solved and COBOL application previously usable only in closed environment was modified to become a portable application. As a result, there is an easily maintainable portable application that has large potential hardware and operating system base. As a result TIPA's life span was increased by 10 or more years, and software product's potential customer base grew substantially. This study reviews the help of software tools and practices to preserve many years of software development and advantage of existing COBOL competence.

ALKUSANAT

Tämä työ on tehty Tietonauha-yhtiö Oy:ssä. Tietonauha on vuonna 1975 perustettu ATK-alan palveluyritys, joka toimittaa palkka- ja henkilöstöhallinnon, materiaalihallinnon ja taloushallinnon järjestelmiä sekä HP3000 -laitteistoille, että avoimeen järjestelmäarkkitehtuuriin. Tietonauhassa oli vuonna 1993 noin 90 ATK-alan työntekijää. Liikevaihto oli vuonna 1993 yli 50 milj. mk.

Työn tarkastajana toimivat professorit Tommi Mikkonen ja Seppo Kuikka, joille haluan esittää kiitokseni saamistani neuvoista ja opastuksesta. Kiitokset myös Tietonauhassa työn ohjaajana toimineelle johtaja Visa-Matti Leinikille sekä oikeinkirjoituksen tarkastaneelle Tekniikan Tohtori Petteri Uusimaalle. Erityisen kiitoksen haluan antaa perheelleni ja Diplomi-insinööri Petteri Pitkäselle, jotka kannustivat minua työn loppuunsaattamiseksi.

Tampereella 16.4.2010

Timo Pokkinen
Tiilikatu 31
33530 Tampere
050 5319 448

SISÄLLYS

1. JOHDANTO

Tietojenkäsittelyyn käytettävät laitteet kehittyvät nopeasti. Samalla tehokkuuden lisääntyessä ovat laitteistojen hinnat pudonneet niin, että ohjelmistot ovat se osa-alue tietojenkäsittelyssä, jonka tehostamiseen kulutettu aika ja raha ovat yrityksien toiminnan kehittämistä rajoittava ja jopa estävä tekijänä.

Ohjelmointityön vakiintumattomuus on usein noussut esiin laitteiden kehittyessä ja ohjelmistojen vaatimuksien ja koon kasvaessa. Uusien ohjelmoinnin apuvälineiden, menetelmien ja työkalujen käyttöönotto on helpottanut uusien, nykyaikaisten ohjelmistojen kehittelyä ja hallintaa, mutta jo tehdyn työn tulokset−valmiit ja käyttökelpoiset ohjelmat−on usein unohdettu. Useiden henkilötyövuosien aikana kehitellyt ja testatut ohjelmat on jätetty usein pelkäksi malliksi uudelle ohjelmistolle siirryttäessä uusien ja tehokkaampien tietokoneiden ja käyttöjärjestelmien käyttäjiksi. Kauan käytössä olleet, hyvin toimivat ohjelmistot ovat ohjelmistotaloille resurssi, jota ei kannata hukata muutettaessa laitekantaa ja ohjelmointityökaluja. Tietonauha-yhtiön TIPA, palkkahallinnon COBOL-ohjelmointikielellä toteutettu järjestelmä, päätettiin siirtää avoimeen laiteympäristöön.

Laitekannan vaihtoon liittyvät koulutukset ja muutokset on helpompi saada sujuvasti läpikäytyä, kun edes yksi ohjelmoijan ympäristön elementeistä, ohjelmisto, pysyy muuttumattomana. Ohjelmistoa käyttävälle yritykselle laitekannan vaihto on monimutkainen operaatio, jonka sujuvuus riippuu pitkälti henkilökunnan myönteisestä suhtautumisesta muutokseen. Henkilökunnan kannalta muutos on helpoin silloin, kun vain käytettävän laitteen ulkonäkö muuttuu. Ohjelmiston ja sen toimintojen pysyminen ennallaan poistaa osaltaan muutosvastarintaa ja koulutustarvetta.

Tässä diplomityössä tutkitaan siirrettävyyden (portability) ongelmaa ja erästä ratkaisua suurehkon COBOL-kielisen ohjelmiston tapauksessa. Työssä pyritään kuvaamaan kaikkien ohjelmistoon vaikuttavien tekijöiden suhdetta ohjelmistoon ennen ja jälkeen ohjelman konvertoinnin. Jotta ohjelmiston muuttaminen olisi kannattavaa, tulee jokaisen ohjelmistoon vaikuttavan tekijän, ohjelmistotalon, ohjelmoijan, laitteiston ja ohjelmiston käyttäjän hyötyä konvertoinnista.

Työn tekemisessä kontribuutiona tehty esimerkki TIPA-ohjelmiston muuttamisesta kuvaa suurehkon HP3000-tietokoneelle ohjelmoidun henkilöstöhallinnan ohjelmiston muuttamista siirrettäväksi ohjelmistoksi. Aikaisemmin vain HP3000-laitteistossa MPE-käyttöjärjestelmässä toiminut HP COBOL II:lla kirjoitettu ohjelmisto toimii muutoksien jälkeen useassa käyttöjärjestelmässä ja useissa sadoissa eri valmistajien tietokoneissa.

Siirrettävyys, perusteet siirrettävyyden tavoittelulle, siirrettävyyden ongelmat ja ratkaisumahdollisuudet käsitellään seuraavassa luvussa. Siirrettävyyden määritelmä ja

tavoitteet käydään läpi ja tutustutaan niihin ohjelmiston siirrossa esiintyviin asioihin, jotka aiheuttavat suurimman osan ongelmista. Ohjelmiston ja ohjelmistoarkkitehtuurin eri elementit ja niiden siirrettävyyden taustat käydään läpi.

Siirrettävyyden työkaluja, joita voidaan käyttää helpottamaan ohjelmistojen siirtoa ympäristöstä toiselle, tarkastellaan yleisesti, tutkitaan mahdollisia ratkaisuvaihtoehtoja ja selvitetään ongelmien välttämistä siirrettävyyden tavoittelussa. Ohjelmiston muokkaus siirrettäväksi ei saa olla itsetarkoitus ja teoriaosuuden lopuksi esitellään ohjelmiston siirrettävyyden vaikutusta ohjelmistojen muihin yleisiin laadun mittareihin.

Työn käytännön osuudessa kuvataan siirrettäväksi muokattava järjestelmä ja sen osat, joita muokkaus koskee. Lähdejärjestelmä ja sen ympäristö esitellään, muokkaukseen käytettävä laitteisto ja sovellukset kuvataan. Varsinaisesta muokkauksesta esitellään ohjelmiston eri osien ja toiminnallisuuksien vaatimat muutokset sekä muokkausmekanismit.

Ohjelmiston muuttaminen siirrettäväksi käydään läpi tietojen siirtojen ja konversioiden osalta jokaisella ohjelmiston osa-alueella. Lisäksi selvitetään parin erikoistoiminnallisuuden, erätöiden ja lajittelun toteutustapaa ja siirrettävyyden erityisongelmia näissä toiminnallisuuksissa. Käytännön osuuden lopussa kuvataan nykyjärjestelmä, joka on siirrettävissä oleva ohjelmisto.

2. SIIRRETTÄVYYS

2.1. Yleistä

Ohjelmiston siirrettävyys tarkoittaa sitä, että ohjelma toimii vähintään kahdella eri tietokoneella samalla tavalla. Mitkä seikat sitten vaikuttavat ohjelmiston toimintaan ja ovat siirrettävää ohjelmaa kirjoitettaessa otettava huomioon?

Yksi käytäntö, jolla siirrettävyyttä ja sen ongelmia tutkitaan, jakaa ongelmat ja ratkaisut Lecarmen (Lecarme, 1989) tapaan seuraaviin alueisiin: Ympäristö, laskenta ja datan siirto. Lecarme käsittelee näitä alueita hyvinkin matalalla tasolla, eli enimmäkseen laitteisto- ja käyttöjärjestelmätasolla.

Toinen jaottelu, jota käytetään, jakaa siirrettävyyden ongelmat ja ratkaisut Hendersonin (Henderson, 1988) tapaan kahteen alueeseen; Laitteisto ja ohjelmointi. Tässä jaossa ohjelmointia on kaikki muu paitsi fyysinen laite käyttöjärjestelmistä lähtien. Tämän jaottelun laitteisto-osa käsittelee samoja asioita, joihin Lecarme ottaa kantaa jaottelunsa pohjalta.

Näiden jakojen erilaisuus johtuu näkökulmasta, joka siirrettävyyteen otetaan. Siirrettävyyden yleisiä ongelmia tutkiva Lecarme katsoo ongelmia kokonaisuutena, jonka toimivuus saavutetaan laitteiden ja ohjelmistojen yhteisen yhdenmukaisuuden tuloksena. Tämä on käytännössä mahdotonta kaupallisesta näkökulmasta, vaikkakin yhteistyötä suurten laitevalmistajien välillä on olemassa.

Seuraavissa luvuissa siirrettävyyttä käsitellään Lecarmen lajittelun pohjalta ja myös ohjelmiston näkökulmasta. Ohjelmistoon kuuluvat ohjelmointiympäristö, eli käyttöjärjestelmä ja varusohjelmistot, ja ohjelmointikieli ja sitä tukevat työkalut.

Näiden lisäksi yksi mainittava siirtotyön käytännön ongelmana, joka on tullut esiin ohjelmistojen siirtoprojekteissa, on tiedonvälityksen ongelma. Eri ihmiset kehittävät, asentavat ja muokkaavat siirrettäväksi ohjelmistoa, jolloin muutosvaiheen tiedonkulun täytyy toimia kitkattomasti. Koska ohjelmiston kehitystä ja ylläpitoa ei voida keskeyttää siirtotyön ajaksi, täytyy siirtotyön ajan olla selkeät rajaukset kehitystyölle. Kehitystyössä tulee pitäytyä tiukasti vallitsevaan ohjelmointityyliin ja käyttää mahdollisimman paljon olemassa olevaa koodia ja yksinkertaista toteutustapaa.

Siirrettävyyden saavuttamiseksi on toteutettu useita akateemisia kokeiluja, joista muutamat ovat yleistyneet kaupallisiksi sovelluksiksi. Laitevalmistajat eivät ole panostaneet näihin työkaluihin, mutta ohjelmistopuolella työtä on tehty markkinaosuuksien ylläpitämiseksi tai kasvattamiseksi. Muuttuvat laitealustat ja käyttöjärjestelmät vaativat siirrettävyyttä helpottavia välineitä. Siirrettävyyden työkalut voidaan jakaa seuraaviin kokonaisuuksiin:

- Ohjelmointikielet, jotka ovat siirrettäviä.
- Ohjelmistokokonaisuuksiin tai kehittimiin, joiden avulla siirrettävän ohjelmiston ympäristö ja liittymä laitteisiin pidetään vakiona.
- Ohjelmiin ja toimintatapoihin, joiden avulla siirrettävää ohjelmakoodia tehdään, koodia muokataan siirrettäväksi tai ympäristön muutokset piilotetaan ohjelmalta.

Siirrettävyys ei saa olla itsetarkoitus. Siirrettävyys tuo mukanaan tärkeitä toimintatapoja ohjelmointiin ja asettaa ohjelmoinnin välineiden valinnalle vaatimuksia, jotka ovat hyvän ohjelmiston rakentamiselle lähes välttämättömiä ja ainakin hyödyksi. Joissain tapauksissa siirrettävyyden tavoittelu heikentää ohjelmiston laatua tai käytettävyyttä siinä määrin, että siirrettävyyden hyödyt jäävät saavuttamatta.

Siirrettävyyden tavoittelun tulokset ovat pitkälti samat mitä vaaditaan laadukkaalta ohjelmistolta. Voidaankin sanoa, että laadukas ohjelmisto on usein siirrettävä ja hyvä siirrettävä ohjelmisto on laadukas, koska se on rakennettu sekä modulaariseksi, että yksinkertaiseksi.

2.2. Perusteet siirrettävyyden tavoittelulle

"Olisi hienoa, jos kaikki ohjelmamme toimisivat kaikilla tietokoneilla"

Yritysten ostaessa laitteet ja ohjelmistot erikseen periaatteella ”Keskuskoneemme on Unix-laite ja kaikilla toimihenkilöillä on PC; Mitkä ohjelmistot niissä toimivat" eivät ohjelmistotalot voi toimittaa omaa laitteen ja ohjelmiston pakettiratkaisua.

Ohjelmistotoimittajan sopeutuminen asiakkaan vaatimuksiin tuo etua sekä toimittajalle että asiakkaalle.

Toiminnot, kuten järjestelmiin sisään kirjoittautuminen, tausta-ajot, tiedostojen nimeäminen, nauhureiden käyttö, oheislaitteet ja sananpituuden rajoitukset vaihtelevat laitteesta toiseen. Käytettävien työkalujen tulisi kuitenkin sopeutua automaattisesti näihin ympäristön muutoksiin ja olla edelleen tehokkaita. Vaikka ohjelmointistandardien mukaan tehty ohjelmisto ei olisikaan täydellisesti siirrettävä, tulisi ohjelmiston toimia eri laitteissa olennaisilta osiltaan samalla tavalla ja toiminnallisesti yhteneväisesti. Tavoitteena ohjelmiston ja ohjelmointimenetelmien standardoinnissa on absoluuttinen siirrettävyys. (Tausworthe, 1981)

Siirrettävä ohjelmisto on toimittajan vahva kilpailuvaltti, koska ohjelmistoa voidaan tarjota asiakkaille riippumatta heidän laitevalinnastaan. Hinnoittelu on varmalla pohjalla, koska eri laiteympäristöjen vaatimat muutokset ovat selkeästi yksilöitävissä. Potentiaalinen asiakaskunta kasvaa. Asiakkaan laiteympäristön muutos ei vaikuta yrityksen työntekijöiden tehtäviin ja muutosvastarintaa ei esiinny, koska käytettävä sovellus pysyy samana. Koulutuskulut jäävät mahdollisimman pieniksi. Ohjelmisto saattaa olla välttämätön yrityksen toiminnan kannalta, jolloin se ei ole esteenä muun tietotekniikan käytön kehittymiselle, eikä rajoita yrityksen muuta toimintaa. Useiden

erilaisten laitteiden ollessa verkossa voidaan samaa ohjelmistoa ja sen tuottamaa dataa käyttää hyväksi kaikissa laitteissa.

Hyvän siirrettävän ohjelmiston kehittäjät voivat keskittyä sovelluksen ominaisuuksien kehittämiseen joutumatta miettimään esimerkiksi eri käyttöjärjestelmien erojen vaatimia muutoksia ohjelmistoon. Kokeneet suunnittelijat voivat käyttää ammattitaitoaan tutuilla työkaluilla tehokkaasti, eikä lisäkoulutusta eri laiteympäristöihin tarvita. Muutokset eri laiteympäristöihin voidaan tehdä nopeasti ohjelmiston tiettyihin osiin, ja ohjelmiston testaus uudessa laiteympäristössä voidaan aloittaa nopeasti.

Kehitettäessä ohjelmistoa siirrettäväksi sitä testataan tehokkaasti, ja näin ollen siirrettävässä ohjelmistossa on vähemmän virheitä. Siirrettäväksi tehdyn ohjelmiston ostava asiakas saa tuotannossa olleen ohjelmiston, joka on kehittynyt muiden asiakkaiden vaatimuksesta paremmaksi ja jossa suurin osa virheistä on jo korjattu. Työpanos, joka suurissa ohjelmistoissa on useita henkilötyövuosia, ei mene hukkaan, ja uusia välineitä voidaan käyttää monipuolistamaan ja parantamaan ohjelmistoa. Ohjelmiston elinikä kasvaa ja siirrettävyyden myötä ohjelmisto on tarkemmin suunniteltu ja modulaarisempi, jolloin muutosten teko on helpompaa.

Ohjelmiston kehitys siirrettäväksi saattaa kuitenkin tuoda mukanaan ongelmia. Ohjelmiston siirto uuteen ympäristöön voi tuoda lisää esimerkiksi laskennallisia virheitä. Ohjelmiston tehokkuus voi pudota ja suorituskykyongelmia saattaa esiintyä. Koodin uudelleenkirjoitus kokonaan toisilla työkaluilla voisi olla parempi vaihtoehto, koska ohjelmisto yleensä paranee huomattavasti uudelleen kirjoitettaessa.

2.3. Ympäristö

Ohjelmiston ympäristöön kuuluvat siirrettävällä ohjelmalla lähtöympäristö, siirtotyön toteutusympäristö ja kohdeympäristö.

Lähtöympäristö on useasti monen käyttäjän ympäristö, josta hyväksi havaittu ohjelmisto siirretään muihin ympäristöihin. Lähtöympäristö voi myös olla yhden käyttäjän ympäristö, jolla oleva ohjelmisto on ominaisuuksiltaan erinomainen tai niin tärkeä, että sen siirtotyö kannattaa. Lähtöympäristön ohjelmointikieli ja työkalut ovat yleensä vanhanaikaisia tai yhteen ainoaan laitteistoon sidottuja.

Siirtotyön toteutusympäristö on monesti joko lähtö- tai kohdeympäristö. Siirtotyön suorittaa yleensä pieni ryhmä tai yksi henkilö, jolloin siirtotyön vaatima laitekapasiteetti on vähäinen. Siirtotyön toteutusympäristön ollessa erillisessä laitteessaan ei tuotantotoiminta eikä muu kehitys häiriinny. Siirtotyön toteutusympäristön työkalut on usein tehtävä itse tapauskohtaisesti, koska valmiita ja sopivia työkaluja siirtotyöhön on hyvin vähän saatavilla.

Kohdeympäristöinä ovat nykyaikaiset, käyttökustannuksiltaan halvat laitteet ja nykyaikaiset ohjelmointivälineet esim. ohjelmistokehittimet. Päätös kohdeympäristöstä on toissijainen ongelma, koska suurin työ tehdään ohjelmiston muokkaamisessa

siirrettäväksi, jonka jälkeen ohjelmiston siirto eri kohdeympäristöihin on selvästi rajattu työ, jonka työmäärä on suhteellisen pieni siirtotyön kokonaistyömäärään nähden.

Kohdeympäristön valinta jo ennen työn aloittamista saattaa rajoittaa työmäärää joltain osin, mutta saavutettu siirrettävyyden taso on tietenkin tämän jälkeen matalampi ja ohjelmiston mahdollinen siirto edelleen kolmanteen ympäristöön saattaa olla työmäärältään jopa suurempi kuin ensimmäisessä siirtotyön toteutuksessa.

Eri ympäristöissä on suuriakin eroja ja siirtotyön ympäristön valinta tulee tehdä siten, että kaikkien haluttujen kohdeympäristöjen eroavaisuudet täytyy pystyä joko poistamaan siirtotyön toteutusympäristössä tai vähintään eroavaisuuksien vaikutukset pitää pystyä minimoimaan. Laitteistojen käskykannat ovat erilaisia, mutta tästä ei ole suurtakaan haittaa siirrettävää ohjelmistoa rakennettaessa, koska eri käskykannoilla saadaan aikaan samat toiminnot.

2.4. Laitteisto

Tiedon sisäinen talletustapa laitteistoissa on toteutettu binäärisesti, mutta tietokoneiden sananpituudet vaihtelevat 8 bitistä 256 bittiin. Tämä vaikuttaa koneen kokonaislukualueeseen ja reaalilukujen esitystarkkuuteen.

Kokonaislukuja kuvataan monilla perinteisesti keskuskoneina käytetyillä tietokoneilla 16 bitillä, joista yksi on varattu etumerkille. Tällöin kokonaislukualueeksi tulee -32767 - +32767. Helpoimmin kokonaislukujen siirrettävyyden takaa käyttämällä lukuja vain tältä väliltä (Wallis, 1982). Negatiivisten lukujen kuvaamiseen on kaksi yleisesti käytettyä tapaa, yhden ja kahden komplementti. Kummassakin tavassa merkitsevin bitti on etumerkki. Kahden komplementti tarkoittaa sitä, että 8 bitin tavu voi kuvata numeroa väliltä -128 - +127. Yhden komplementissa vastaavasti -127 - +127. Tällöin on olemassa "negatiivinen nolla" eli kaikki bitit ykkösiä ja kaikki bitit nollia ovat molemmat nollia. Kokonaislukuoperaatioissa tulisi jokaisella tietokoneella olla tieto ylivuodosta. Ylivuototapauksessahan, jollei sitä tunnisteta, jää kokonaisluku epämääräiseen tilaan ja koska kokonaisluvuilla kaikki bittiyhdistelmät ovat useimmiten käytössä, ovat tulokset lukuina hyväksyttäviä. Koska kaikki koneet eivät tunnista ylivuotoa, tulisi lukujen järkevyystestaus suorittaa ohjelmallisesti. Esim. COBOL-koodissa voidaan testata laskutoimitusten ylivuototilanteita, kun tietokoneessa itsessään ylivuoto havaitaan.

Useissa tietokoneissa on käskyt, joilla suoritetaan pitkien kokonaislukujen laskenta ja myös kerto- ja jakolaskut voidaan suorittaa yksittäisellä tietokoneen käskyllä. Kokonaislukujen jakolasku on tällöin sallittua ja oikeellista vain positiivisilla luvuilla. Negatiivisten lukujen jakolasku saattaa tuottaa eri koneilla erilaiset vastaukset (Henderson, 1988).

Eräs yksittäinen ongelma on satunnaislukugeneraattori. Satunnaislukugeneraattorit käyttävät usein koneen lukualueen rajoilla olevia lukuja. Näin ollen pienen lukualueen omaavien laitteiden satunnaisluvut ovat vähemmän satunnaisia.

Lecarme tavoittelee laitteen ominaisuudet piilottavaa yleistä välikieltä, joka olisi siirrettävä laitteistoarkkitehtuurista riippumatta. Rekisterien käyttö tietokoneissa vaihtelee ja useimmissa koneissa tavalliset operaatiot ottavat jonkun parametrinsa rekistereistä tai muistista. Vain pinon käyttö laskutoimitusten välitulosten talletukseen takaa, että laitteissa voidaan käyttää välikieltä, joka olisi yleisesti siirrettävää. Koska koneen sisäisessä tiedonkäsittelyssä on yksinkertaisempi simuloida pinokäsittelyä rekisterejä käyttävissä koneissa kuin toisinpäin, tulisi siirrettävän välikielen käyttää mallina pinokonetta. (Lecarme, 1989)

2.5. Laskenta

Reaaliluvut voidaan tietokoneissa kuvata vain tietyllä tarkkuudella. Tarkkuus riippuu koneen sananpituudesta. Kaikki reaalilukujen operaatiot antavat vain likiarvon oikeasta vastauksesta, ja näin ollen mitä useampia laskutoimituksia tarvitaan, sitä epätarkempi likiarvo saadaan vastaukseksi.

Reaalilukujen kuvaamiseen käytetään useita tapoja. Reaalilukujen osat eksponentti ja mantissa ovat kumpikin etumerkillisiä lukuja, joilla on oma kokonsa. Eksponentin pituus määrää käytettävissä olevan lukualueen koon ja mantissa tarkkuuden. Useasti reaalilukuja voidaan esittää yksin- ja kaksinkertaisella tarkkuudella. Kaksinkertaisella tarkkuudella esitettävien lukujen kaikkia bittejä ei aina ole käytössä, joten esimerkiksi eri laitevalmistajien 48-bittisten lukujen tarkkuus ei välttämättä ole sama. Numeeristen ohjelmistojen siirrettävyydelle reaalilukujen tarkkuus on todellinen ongelma. Ohjelmakirjastot, kuten Fortranin NAG, jonka on suunnitellut englantilainen Numerical Algorithm Group, tarjoavat laskentapainotteisiin ohjelmistoihin siirrettäviä ratkaisuja (Numerical Algorithms Group, 2010).

Koska Fortranin standardointi on puutteellista, on kieleen syntynyt useita ominaisuuksia, jotka ovat kääntäjäkohtaisia. Adassa standardiominaisuutena voidaan ohjelmassa määrittää lukualue ja tarkkuus, jolloin ei tarvitse tietää tiedon esitystapaa tietokoneessa vaan kääntäjälle annetaan tiedon esitystarkkuus. Tällöin numeeriset ohjelmat ovat helpommin siirrettäviä.

Siirrettävyyden mahdollistamiseksi on Wallisilla olemassa ratkaisu. Laskenta tehdään luvuilla, joiden lukualue on +-10 potenssiin 68 ja tarkkuus enintään 6 desimaalia. (Wallis, 1982)

2.6. Tiedonsiirto

Tiedonsiirto eri tietokoneiden välillä on usein välttämätöntä. Sama ohjelmisto voi olla usealla erilaisella laitteella, joilla on mahdollisesti erilainen käyttöjärjestelmä. Tästä syystä ohjelmiston tuottaman datan tulisi olla laiteriippumatonta. Ensisijaisesti dataa tulisi siirtää ainoastaan merkkimuotoisena. Numerotiedon siirto binäärisenä johtaa helposti virheisiin laitteiden sisäisen datan muodon erilaisuuden takia (Wallis, 1982).

Tekstimuotoisen datan ja peräkkäistiedostojen siirto ei tuottane ylitsepääsemättömiä ongelmia, mutta rakenteelliset tietovarastot, listat ja indeksoidut tiedostot ovat vaikeasti siirrettävissä. Tietokannat ovat fyysisesti organisoitu tiettyyn ympäristöön ja sen riippuvuus laitteistosta tekee siitä lähestulkoon mahdotonta siirtää (Lecarme, 1989).

Datan siirrossa on kolme mahdollista muuttuvaa tekijää, laitteistoarkkitehtuuri, käyttöjärjestelmä ja ohjelmointikieli. Näistä yksi tai useampi tekijä voi muuttua siirrossa, mutta useimmiten muuttuvat tekijät, jotka aiheuttavat ongelmia, ovat (Schulmeyer, 1990):

1. Yhteinen laitteisto ja käyttöjärjestelmä, mutta muuttuva ohjelmointikieli.
2. Yhteinen ohjelmointikieli, mutta muuttuva laitteisto ja käyttöjärjestelmä.

Tiedon suojaus ja varmistus siirron yhteydessä tulee muistaa aina, kun siirretään arvokasta tai salaista tietoa. Varmuuskopiot siirrettävästä tiedosta ja siirrettävän tiedon kahdentaminen riittävät useimmissa tapauksissa estämään tiedon katoamisen. Salasanat ja käyttäjätunnukset laitteissa ja tiedon luvussa estävät salaisen tiedon paljastumisen. Tiedon kryptauksessa tai muissa sekoituksessa tulee muistaa, että tiedonsiirto saattaa vaikuttaa muun kuin yksinkertaisen tekstimuotoisen datan käsittelyssä. Tiedon suojaus on kuitenkin joskus välttämätöntä, ja suojauksen hyödyt ja rasitteet tulee punnita tarkkaan.

Tiedonsiirrossa on lisäksi kaksi erilaista ongelmakohtaa, merkkilajistot ja tiedonsiirtomekanismi.

2.6.1. Merkkilajistot

Merkkilajistot ovat pieni ja yksinkertainen asia, josta on yksityisten (IBM), kansallisten (ANSI, IEEE, Afnor) ja kansainvälisten (ISO, CCITT) standardien myötä saatu sekoitus, jonka selvittäminen on kansallisia merkkejä käyttävissä maissa suuri ongelma. Skandinaaviset merkit (Å, å, Ä, ä, Ö, ö) ovat tutuimpia ongelmamerkkejä suomenkielisiä ohjelmistoja ja suomenkielistä dataa käsitteleville käyttäjille ja ohjelmoijille. Eri laitteiden ja varusohjelmien tukemat merkistöt ja niiden muunnokset saadaan usein selville parhaiten kokeilujen kautta.

Standardeista laajimmalle levinnyt on 7-bittinen ASCII-merkistö. 7-bittisessä ISO-merkistössä on kansallisille merkeille omat paikkansa. Suomessa kansalliset merkit aiheuttavat ongelmia, jotka on parasta ratkaista rakentamalla ohjelmistojen tiedonsiirron rajapintaan konversiot merkkilajien välillä. Dataa lähetettäessä tulisi merkkilajisto olla mukana ohjaustietona. ISO:n 8-bittinen standardi voisi yleistyessään tuoda helpotusta merkkilajistojen ongelmaan.

Ohjelmistokehittimissä merkkilajikäsittelyt voidaan parhaiten hoitaa määrittelemällä järjestelmiin oletusmerkkilajit kaikkiin eri tiedonsiirron rajapintoihin. Tulostuksessa voidaan käyttää kirjoitinkohtaista merkkilajistoa ja tiedostoon kirjoitus voi käyttää omaa merkkilajistoaan.

2.6.2. Tiedonsiirtomekanismit

Mekanismit tiedonsiirrossa ovat lähes yhtä monimuotoisia kuin merkkilajistotkin.

Tiedonsiirto listauksilla, reikäkorteilla tai paperinauhalla on jäänyt historiaan muutamia erikoistapauksia lukuun ottamatta. Levykkeet, nauhat ja tiedonsiirto elektronisesti ovat yleistyneet ja siirrettävät tietomäärät ovat kasvaneet valtavasti. Tiedonsiirto on kehittynyt nopeasti vastaamaan uusia tarpeita ja tästä johtuen uusia, tehokkaampia levyjä, nauhoja ja tietokoneverkkoja on syntynyt. Ongelma tiedonsiirrossa voi muodostua yhteisen mekanismin löytymisestä. PC-koneissa on perinteisesti käytetty erilaisia levykkeitä ja suuremmissa laitteissa nauhoja.

Tiedon muoto tietovälineellä on seuraava ongelma. Rivinvaihdot, tiedoston loppumerkit ja muut erikoismerkit vaihtelevat eri käyttöjärjestelmissä ja laitteissa.

Tietokoneverkot ja niissä toimivat tiedonsiirto-ohjelmistot ovat hyvä ratkaisu tiedonsiirtojen ongelmiin. Jos tietokoneiden verkottaminen on mahdollista, löytyy varmasti myös ohjelma, jolla dataa voidaan siirtää laitteelta toiselle. Tällöin käyttöjärjestelmien tiedostomuotojen erilaisuudesta huolehtii tiedon siirtävä ohjelma. Jos laitteita ei voida verkottaa, on paras ratkaisu yleiskäyttöinen tiedonsiirron hallintaohjelmisto, joka tuntee rajatun määrän käyttöjärjestelmiä, tiedostomuotoja ja merkkilajistoja. Tiedonsiirto hoidetaan tällöin erillisillä yhteydenotoilla lähde- ja kohdelaitteisiin.

Erilaiset tiedon kuljetuspalvelut käyttävät useita sovitettuja tiedonsiirtoprotokollia, tai käyttävät jonkun laitetoimittajan de facto -standardia. Tiedonsiirtoon voidaan hallintaohjelmistoa, esim. Frends II:sta, käytettäessä määritellä laitteiden levyiltä omat alueensa, joita hallintaohjelmisto tutkii ja lähettää tiedon muuntaen sen kohdekoneen tuntemaan muotoon.

Tiedonsiirron sisäinen muoto on määritelty tarkasti OVT-standardeissa, joita Suomessa ylläpitää Tiedonsiirtoyhdistys. OVT-malli eli malli organisaatioiden väliseen tiedonsiirtoon sisältää kaikki tarvittavat osatekijät organisaatioiden välisen tiedonsiirron automatisoimiseksi yhteensopivalla tavalla (Marttila, 1990). Siirtyminen organisaatioiden välisessä tiedonsiirrossa EDIFACT-standardin mukaiseen esitystapaan ja tietosisältöön voidaan yleistää myös organisaation sisäiseen tiedonsiirtoon, jolloin tieto eri laitteiden ja järjestelmien välillä siirretään EDIFACT-muodossa. OVT:n käyttöönotto vaatii kuitenkin aina paljon suunnittelua ja koordinointia. Käytettäessä tiedonsiirtoon ulkoisia kuvauksia syntyy aina useampia rajapintoja ja muunnoksia, joihin liittyy oma virhemahdollisuutensa.

2.7. Käyttöjärjestelmät

Käyttöjärjestelmät piilottavat laitteen ominaisuudet toteuttamiensa palveluiden taakse ja tarjoavat sovellusohjelmille liittymän näihin palveluihin (Henderson, 1988).

Käyttöjärjestelmät voivat olla yhden käyttäjän, usean käyttäjän, reaaliaikaisia, eräpohjaisia tai näiden yhdistelmiä. Käyttöjärjestelmien tarjoamista palveluista

oheislaitteiden hallinta on yksi tärkeimmistä sovellusohjelmistojen kannalta. Tiedon siirron, syötön ja talletuksen eri muodot ovat olennainen osa sovelluksen käyttöä.

Yksinkertaisimmillaan käyttöjärjestelmä tarjoaa rajoittamattoman mahdollisuuden käyttää laitteen levyä. Seuraavalla tasolla tiedon täytyy olla jo jollain tietyllä alueella levyllä ja monimutkaisimmillaan tiedonhallinta käsittää oikeuksien hallinnan, tiedon muodon hallinnan, tiedon sisällön osittaisen hallinnan, tiedon nimeämiskäytännöt ja tiedon käsittelyn rajaukset. Jokainen uusi palvelu tai rajoite saattaa olla esteenä siirrettävyydelle. Esimerkiksi tiedoston nimi voidaan rajoittaa 6,8,32 tai useampaan merkkiin. Yhtäaikaisten avointen tiedostojen määrä on myös eräs käytännössä esiin tuleva ongelma.

Käyttöjärjestelmät ovat paisuneet uusien ominaisuuksien mukaan ja yhtenäisyys komennoissa ja niiden syntaksissa ja semantiikassa on usein hävinnyt. Käyttöjärjestelmien ominaisuuksissa on suuria eroja ja ne ovat niin monimutkaisia, ettei niitä pysty analysoimaan (Lecarme, 1989). Käyttöjärjestelmän komennot, joita joissain ohjelmistoissa joudutaan käyttämään, ovat erilaisia jopa saman käyttöjärjestelmän eri versioissa. Unix -käyttöjärjestelmät ovat kehittyneet eri valmistajilla ajan myötä entistä enemmän erilleen, eivätkä pyrkimykset standardiin Unixiin ole johtaneet merkittävän yhtenäistymiseen. Esimerkiksi tulostuskomento *lp*:n optiot vaihtelevat siten, että samalla optiolla on eri merkitys.

Jotkut käyttöjärjestelmät vaativat resurssien varaamista ennakolta, kun taas toisissa resurssien jako tapahtuu dynaamisesti. Jotta järjestelmä olisi siirrettävä, sen tulisi pystyä ennakolta varaamaan oletettu määrä resursseja käyttöönsä ja sillä tulisi olla mekanismi selviytyä resurssien loppumisesta. Tämä resurssien varaus näkyy toimintojen hidastumisena joissain toimenpiteissä ja kasvattaa ohjelman kokonaisresurssitarvetta. Optimaalista olisi olla käyttämättä mitään käyttöjärjestelmän resursseja, mutta se ei ole suorituskyvyn kannalta toteutuskelpoista.

Virhetilanteiden käsittely vaihtelee käyttöjärjestelmästä toiseen ja niiden hallinta sovelluksissa tulisi tapahtua aina omassa moduulissaan.

Suljetuissa järjestelmissä esim. AS400 ja HP3000, varusohjelmistot ja yleisesti käytetyt tehokkaat työkalut houkuttelevat niiden käyttöön sovelluksista. Nämä palvelut ovat myös usein helposti liitettävissä sovelluksiin, mutta ne eivät ole lainkaan siirrettäviä. Käyttöjärjestelmäkomennot ja varusohjelmistokutsut tulisi eristää ohjelmista erilliseksi ylläpidettäväksi osaksi ohjelmistoa ja sovellusten käyttämät käyttöjärjestelmäpalvelut tulisi olla voimakkaasti parametroitavissa.

2.8. Korkean tason ohjelmointikielet

Ohjelmointikielten välisestä paremmuudesta on kiistelty pitkään. Ohjelmointikielten paremmuus ratkeaa kuitenkin aina käyttötarkoituksen mukaan. Siirrettävyyden saavuttamiseksi ei voida kirjoittaa ohjelmaa konekielisenä tai assemblerilla. Siirrettävyys tuntuisi siis vaativan korkeamman tason ohjelmointikielen.

Korkean tason ohjelmointikielillä on muutamia siirrettävyyden kannalta merkittäviä ominaisuuksia, kuten ympäristöriippumattomuus, laitteistoriippumattomuus, käyttöjärjestelmäriippumattomuus (Lecarme, 1989). Kaikki nämä ominaisuudet tosin vaativat kääntäjien tai muiden ohjelmistojen tukea resurssien käyttöönotossa.
Ohjelmointikielen valintaan vaikuttavat Hendersonin mukaan (Henderson, 1988) seuraavat seikat:

1. Ohjelmointikielen tuntemus.
2. Saatavuus.
3. Sopivuus.
4. Standardointi.

Kaikilla kielillä on omat hyvät puolensa, mutta valinta täytyy tehdä. Valinta todennäköisesti kohdistuu johonkin olemassa olevaan kieleen ja olemassa olevaan kääntäjään, kun toinen vaihtoehto on tehdä itse ohjelmointikieli ja kääntäjät tarvittaviin ympäristöihin. Olemassa olevia käytettäviä kieliä ovat aiemmin olleet esim. COBOL, BASIC, Pascal, C ja Lisp. Nykyään käytettävistä siirrettävistä ohjelmointikielistä Java, Python ja Ruby ovat laajalti käytössä ja niissä on huomioitu paljon siirrettävyyteen liittyviä ominaisuuksia.

Näissä olemassa olevissa ohjelmointikielissä ongelmaksi tulee niiden muuttuminen ja kehittyminen. Lisäominaisuuksia kieleen tulee jatkuvasti ja niitä standardoidaan myöhäisemmässä vaiheessa. Vanhemmat kääntäjät ja tulkit eivät kuitenkaan tue uusia standardeja ja uudet kääntäjät tuovat taas kieleen uusia ominaisuuksia, joita on vaikea välttää. Standardointi kääntäjien osalta pitää olla riittävää ja kääntäjien toiminnan tarkastamiseen tulisi olla testiohjelmat, jotka jokainen kääntäjä käsittelee samalla tavalla. Näin saadaan aikaan kääntäjien yhteensopivuus, joka on välttämätön ohjelmien yhteensopivuudelle.

Toinen ohjelmointikielten ryhmä koostuu yksittäisten toimittajien toimittamista kielistä. Jotkin näistä voivat olla hyvinkin suosittuja ja toimivia. Näillä kielillä on yhteistä se, että kääntäjät, joita tekee tässä tapauksessa vain yksi toimittaja, ovat yhteensopivia. Näitä kieliä ovat sekä yksittäiset ohjelmointikielet, että versiot ”standardikielistä” (Henderson, 1988).

Yksi ohjelmointikielten valinnassa huomioitava seikka on, että ohjelmointikieli tukee modulaarisuutta ja moduulit voidaan kääntää yksittäin. Tämä helpottaa muutoksien eristämistä tiettyihin moduuleihin ja varmistaa suoraan siirrettävien osien muuttumattomuuden.

3. VÄLINEITÄ SIIRRETTÄVYYDEN SAAVUTTAMISEKSI

3.1. Ohjelmointikielten apuvälineet

Korkean tason ohjelmointikielten standardin mukaisen ohjelmoinnin tarkastamiseen on muutamia välineitä.

Suodatin (filter) jollekin kielelle on ohjelma, joka tarkastaa, ettei koodissa esiinny standardoimattomia ohjelmarakenteita. PFORT on eräs FORTRAN:lle tehty suodatin. PBASIC, BASIC:n murre on PFORT:lla suodatettavaa koodia. Suodattimet tekevät koodin tarkastuksen, kuten kääntäjät, mutta eivät generoi edes välikieltä. Suodattimet kuitenkin usein tarkastavat koodin täsmällisemmin kuin useat kääntäjät.

Tarkastaja (verifier) tutkii ohjelman muuttujien käytön ja ohjelman dynaamisen toiminnan. Esimerkiksi alustamattomien muuttujien ja rakenteiden sisäisten muuttujien käyttö niiden ulkopuolella tarkastetaan, jos näitä ei standardissa hyväksytä. FORTRAN:in dynaamisen tarkastelun tekee esimerkiksi DAVE. (Lecarme, 1989).

Esiprosessori (preprosessor) muuttaa koodin kielen toiseen murteeseen tai kokonaan toiselle kielelle. FORTRAN:lle on olemassa useita esiprosessoreja, esimerkiksi RATFOR ja IFTRAN.

Nämä kaikki tekniikat tutkivat käytännössä koodia merkki merkiltä ja ovat siten hitaita; yksinkertainen esiprosessori, joka tekee vain pieniä muutoksia koodiin, voi olla yhtä hidas kuin tuloksena syntyvän tekstin kääntäminen (Wallis, 1982).

3.2. Siirrettävät käyttöjärjestelmät

Siirrettävät käyttöjärjestelmät piilottavat ohjelmakoodilta kaikki laitteiston ominaisuudet. Siirrettävät korkean tason kielien kääntäjät ovat mahdollistaneet sen, että siirrettäviä käyttöjärjestelmiä on alettu suunnittelemaan ja toteuttamaan. Nämä käyttöjärjestelmät ovat muuttaneet siirrettävien ohjelmien rakentamisen ongelmat kokonaan pois sovelluksista, koska kaikki samat toiminnot saadaan aikaan samoilla käskyillä. Siirrettävät käyttöjärjestelmät pyrkivät luomaan abstraktin ”välikoneen”, joka toimii kaikilla laitealustoilla samalla tavalla. Eri laitteiden rajoitukset tosin tekevät siirrettävän käyttöjärjestelmän tehottomaksi joillakin laitteilla, koska laitteiden ominaisuuksia ei pystytä käyttämään ilman suuria muutoksia käyttöjärjestelmän koodissa.

Ensimmäiset siirrettävät käyttöjärjestelmät olivat kokeiluluontoisia ja siirrettävyyden saavuttamiseksi jouduttiin tekemään työtä lähestulkoon yhtä paljon kuin

sovelluksien uudelleenkoodaamiseen. Näitä ensimmäisiä tulkkaavia, erittäin rajoitetusti siirrettäviä käyttöjärjestelmiä olivat esim. OS6, TRIPOS ja SOLO.

Seuraavat yritykset olivatkin jo menestyksekkäämpiä, vaikka toiset niistä ovat jo unohtuneet. Unix tehtiin PDP-11 laitteelle C-ohjelmointikielellä ja sen ensimmäisissä versioissa siirto toiselle laitteelle onnistui muuttamalla vain 5 % käyttöjärjestelmän koodista. THOTH kehitettiin ”tyhjän koneen” käyttöjärjestelmäksi, jonka avulla laitteeseen voitaisiin sitten asentaa lopullinen käyttöjärjestelmä ”half-bootstrap”-menetelmällä. THOTH oli kaupallinen tuote, joka ei kuitenkaan menestynyt, vaikkakin minikoneympäristössä kaikki kokemukset siitä olivat hyviä.

MUSS on kehitetty Manchesterin yliopistossa kymmenen vuoden aikana ja sitä on käytetty pienistä minikoneista aina Mainframe laitteille asti. Projektin on havaittu hyvällä suunnittelulla ja tarkalla toteutuksellaan jopa ylittäneen toiveet ja sen tuloksena järjestelmien ylläpidon kulut ovat pienentyneet huomattavasti (Wallis, 1982).

Mikrotietokoneilla menestyneimpiä siirrettäviä käyttöjärjestelmiä ovat olleet Microproducts Softwaren MicroCOBOL ja USDC Pascal-järjestelmä. MicroCOBOL on kokonainen ohjelmointiympäristö, jossa on suuri määrä tuotteita kaupallisten ohjelmistojen luomiseen. MicroCOBOL perustuu virtuaalilaitteeseen, jonka luominen eri laitteistoille on hyvin dokumentoitu. Virtuaalikoneen teko aloitetaan Assembly-kielellä ja luomalla ajurit. Tämä osa onkin työläin osa, jonka jälkeen toiminnallisuuden tarkastaminen voidaan automatisoida. USDC Pascal-järjestelmä on tulkkaava ympäristö, jolle on järjestelmiä tukevia tuotteita ja FORTRAN, BASIC ja tietysti Pascal-kääntäjät.

Mikrotietokoneen käyttöjärjestelmä DOS ja myöhemmin Windows ovat saavuttaneet suunnattoman suosion. Microsoft on sen suurin toimittaja ja se on luotu Intel 8086 ja yhteensopiville prosessoreille. IBM PC:n suuren suosion jälkeen PC-arkkitehtuurista tuli de facto-standardi, kuten käyttöjärjestelmästäkin (Henderson, 1988). DOS-ohjelmien siirto laitteilta toiselle on useasti ongelmallista, mutta rajoitukset ovat tunnettuja. Windowsin osalta tilanne on samankaltainen ja ongelmana on käyttöjärjestelmän nopea muutostahti.

3.3. Uudelleenkirjoitus

Koodin uudelleenkirjoitus joko toisella ohjelmointikielellä tai saman ohjelmointikielen siirrettävällä versiolla on yksinkertaisin tapa siirrettävyyden saavuttamiseksi. Uudelleenkirjoitus on suhteellisen edullista, jos vanha järjestelmä on hyvin dokumentoitu. Uuden version hinta on noin 20 prosenttia alkuperäisestä kustannuksesta (Brown, 1977).

Uudelleenkirjoituksen etuna on mahdollisuus muuttaa tunnettuja heikkoja kohtia ohjelmistosta. Ohjelmiston uudelleenkirjoittaminen voidaan tehdä alkuperäisen ohjelmiston kirjoittaneessa yrityksessä, mutta sitä on vaikea toteuttaa muualla. Jos vanhan ohjelmiston koodaajat voivat ottaa osaa työhön, saadaan lisää nopeutta ja dokumenttien puutteellisuudesta ei ole haittaa samassa määrin kuin ohjelmistoa

huonosti tuntevilla koodaajilla. Uudelleenkirjoitus vaatii koodaajilta paljon ja suurten ohjelmistojen uudelleenkirjoitukseen on vaikea saada resursseja.

3.4. Emulointi

Emulointia käytetään, jotta alun perin yhdelle laitteelle suunniteltu ohjelmisto voidaan muokata siirrettäväksi.

Historiallisesti suosituilla laitteistoilla on useita suuria ohjelmistoja, jotka ovat hyödyllisiä. Ohjelmisto halutaan toimivaksi toisessa käyttöjärjestelmässä tai laitteessa. Laitteistotoimittajat eivät enää usko asiakkaidensa vaihtavan ohjelmistojaan täydellisesti aina, kun laitteistosta ja varusohjelmista tulee uusi versio. Emulointi voidaan saada aikaan kahta kautta, laitteistolla tai ohjelmistolla (Henderson, 1988).

3.4.1. Laitteistolla emulointi

Laitteistot voidaan ohjelmoida toimimaan eri tavoin mikrokooditasolla. Tehokkuus ei ole samaa luokkaa kuin puhtaasti laitteistotasolla tehty koodaus, mutta tämä ei ole ongelma kuin tehokkaimmissa suurkoneissa. Useimmiten mikrokoodi on ROM-muistissa, josta se käynnistetään, kun koneeseen kytketään virta. Emulointiohjelmien koosta riippuen niitä voi yhdellä laitteella olla useitakin. Vaikka kahden järjestelmän mikrokoodit ovatkin samassa prosessorissa, tulee näiden välille rakentaa linkit vanhasta järjestelmästä uuteen, koska tuskinpa kaikki vanhan järjestelmän käskyt ovat suoraan uuden järjestelmän käskyjen osia tai täysin samanlaisia.

Yksi esimerkki on Z80-kortti, joka voidaan asentaa Apple II:lle. Applesta tulee CP/M-järjestelmä ja Z80 käyttää muistia ja I/O:ta kuten alkuperäisellä laitteella. Z80-prosessorille voidaan ohjata tietyt toiminnot. Tässä toteutuksessa hyvä puoli on sen halpuus. Prosessori maksaa vähemmän kuin ohjelmien muuttaminen siten, että emulointi toteutuu. Ohjelmien suorituskyky ei laske ja prosessorin toiminta takaa ohjelmien toimivuuden.

3.4.2. Ohjelmistolla emulointi

Ongelmat käyttöjärjestelmän emuloinnissa ovat pitkälti samat kuin laitteistolla emuloinnissa. Uuden järjestelmän tulee toteuttaa kaikki vanhan käyttöjärjestelmän toiminnot. Uudessa järjestelmässä saattaa vanhan järjestelmän toimintojen toteuttaminen olla vaikeampaa kuin uuden järjestelmän toiminnon toteutus on alun perin ollut.

Yksi ratkaisu näihin ongelmiin ja joidenkin käyttöjärjestelmätoimintojen täydelliselle puuttumiselle on ”run-time”-lähestyminen. Tämä tarkoittaa sitä, että vanhan käyttöjärjestelmän kutsut ohjataan ”run-time”-järjestelmälle, joka tulkitsee kutsut ja rakentaa niistä uuden käyttöjärjestelmän kutsuja. Tällöin emulointia ei tarvitse koodata uuteen käyttöjärjestelmään.

Toinen ratkaisu on toteuttaa kaikki vanhan käyttöjärjestelmän kutsut uusien toimintojen osana. Tällöin uuteen käyttöjärjestelmään täytyy jo suunnitteluvaiheessa ottaa huomioon vanhan käyttöjärjestelmän vaatimukset ja tukea niitä.

Emulointi aiheuttaa aina tehokkuuden heikkenemistä ja toisen ratkaisun, jatkuvan alaspäin yhteensopivuuden seurauksena järjestelmien kehitys vaikeutuu. Emulointi on erityisen selkeästi esillä upotettujen järjestelmien luonnissa ja testauksessa, kun järjestelmän seurantaa ei esimerkiksi pesukoneen ohjelmakoneistossa voida toteuttaa.

3.5. Makroprosessorit

Makroprosessorin määritelmät eivät ole kirjallisuudessa ole aivan yhteneviä, mutta kaikki makroprosessorit korvaavat tekstiä. Makroprosessori tutkii lähdekoodia ja korvaa makrokutsut käytössä olevilla komentojonoilla. Esimerkkinä voidaan antaa tyypillinen makromäärittely (Lecarme, 1989).

```
$MACRODEF
header
macro body
$MACROEND
```

Mallissa header eli makron otsikko on merkkijono, joka voi sisältää erikoismerkkejä parametrien määrittelyä varten. Otsikkoa verrataan lähdekoodiin korvausvaiheessa. Makron parametrointi voi vaihdella parametrien puuttumisesta (pelkkä merkkikonversio) aina rekursiivisiin makrokutsuihin. Makron otsikko-osan kuvaus voi olla muotoa

```
SUM $1,$2,$3
```

jossa $n on parametri tai muuttuja.

Makron runko on merkkijono, joka kirjoitetaan otsikon tilalle korvausvaiheessa. Makron runko voi olla otsikon tapaan monentyyppinen. Makrossa voidaan pelkästään korvata merkkijono toisella tai kutsua toisia makroja. Esimerkin runko voisi olla käytettyjen parametrien käsittely muodossa

```
FETCH $1
ADD $2
STORE $3
```

Korvausvaiheessa, makroa käytettäessä, annettu kutsu

```
SUM P,Q,R
```

korvataan makron mukaan komentojonolla

```
FETCH P
ADD Q
STORE R
```

Makroprosessorien rekursiivisuus, kutsujen monipuolisuus ja erilaisten muuttujien asetus- ja testausmahdollisuus korvausvaiheessa ovat ominaisuuksia, joilla yhdessä makrossa tehty valinta voi vaikuttaa toisen makron korvaukseen myöhemmin. Nämä

ominaisuudet tekevät makroprosessoreista erittäin tehokkaita. Makroprosessorit voivatkin tehdä kaiken, minkä kääntäjät tekevät (Wallis, 1982).

Makroprosessoreita voidaan käyttää esiprosessoreina jollekin ohjelmointikielelle. Syntaktiset makroprosessorit ovat osa kääntäjää ja niitä voidaan käyttää lisäämään käännettyyn koodiin erilaisia rakenteita (Lecarme, 1989).

Siirrettävyydelle makrokutsut ovat yksi työkalu, jolla ohjelmiston laitteistoriippuvaisuudet voidaan eristää ja eri laitteistoilla korvata niille sopivilla makroilla. Siirrettävyys johonkin toiseen ympäristöön vaatii tällä tekniikalla siis ainoastaan makrojen uudelleenkirjoittamisen.

3.6. Muuntimet

Muunnin muuntaa korkean tason kieltä toiseksi korkean tason kieleksi tai mahdollisesti jopa takaisin samaksi kieleksi. Toiseksi korkean tason kieleksi muuntaminen voi olla tarpeen, kun ohjelma halutaan toiselle laitteelle, jolle ei kuitenkaan löydy kääntäjää tälle kielelle. Samaksi kieleksi muuntamisen perusteena voi olla erilaisten koodien yhdenmukaistaminen.

Esimerkki ADA:n ja Pascalin välisestä muunnoksesta selvittää hyvin muuntamisen tekniikan (Lecarme, 1989). Menetelmässä luotiin ADA:sta ja Pascalista yhteensopivat käskyjoukot, joiden avulla muunnos kielestä toiseen voisi tapahtua näiden rajoitettujen käskyjoukkojen kautta. Käskyjoukot nimettiin PascalA:ksi ja ADAP:ksi. Lisäksi luotiin laajennukset näihin käskyjoukkoihin, ADAPE ja PascalAE, jotka ovat edelleen osajoukkoja alkuperäisistä kielistä. Laajennukset voidaan kuvata rajoitettujen käskyjoukkojen avulla. Käskyjoukoille luotiin puumalli (tree), jota käytetään useimmissa muunnoksen vaiheissa. Muunnoksessa on alla olevien kuvausten mukaiset kahdeksan komponenttia.

1. Pascal (source) -> PascalAE (tree), tukemattomien rakenteiden poisto.
2. PascalAE (tree) -> PascalA (tree), rakenteiden muunnos.
3. PascalA (tree) -> standardi puumalli, tukemattomien rakenteiden poisto.
4. Standardi puumalli -> Pascal (source).
5. ADA (source) -> Ada (tree), kaikkien rakenteiden hyväksyntä.
6. ADAPE (source) -> AdaP (tree), rakenteiden muunnos.
7. AdaP (tree) -> standardi puumalli, tukemattomien rakenteiden poisto.
8. Standardi puumalli -> ADA (source).

Käyttämällä eri komponentteja saadaan Pascal-koodeista (PascalAE) Pascalin kanssa yhteensopiva ADA-koodi (ADAP). Mikään komponenteista ei ole ylivoimaisen monimutkainen, mikä saadaan aikaan yhteensopivien käskyjoukkojen rajaamisella.

Tavoitteena ei muunnoksessa olekaan monipuolisesti koodattu kohdekoodi, vaan on tyydytty pelkkään ohjelmointikielen vaihtumiseen.

3.7. Kääntäjät

Kääntäjä on ohjelma, joka tuottaa ohjelmointikielestä lähdekielisen ohjelman ajettavan version, objektikoodin.

Määritelmän kuvaama toiminto voidaan saada aikaan monella tavalla. Voidaan käyttää kääntäjää, joka tekee korkean tason ohjelmointikielestä suoraan ajettavaa objektikoodia. Kääntäjä voi tehdä korkean tason ohjelmointikielestä myös matalan tason ohjelmointikieltä, esimerkiksi assembleria. Kääntäjä, joka tekee välikielistä, abstraktin laitteen objektikoodia, tarvitsee tulkin, joka ajaa välikielisen koodin kohdelaitteistossa.

Ohjelmien siirrettävyyden saavuttamiseksi on mahdollista tehdä kääntäjästä siirrettävä. Tämä tuntuu aluksi raskaalta tavalta saavuttaa siirrettävyys, mutta oikeilla menetelmillä kääntäjän siirto uuteen laitteistoon antaa siirrettävyyden verrattain helpolla tavalla. Kuten makroprosessoreiden tapauksessa, on myös kääntäjien siirrettävyyttä tutkittaessa on selkeästi rajattava ne osat, jotka ovat laitteistoriippuvaisia. Siirrettävien kääntäjien suosio on selkeästi suurempi kuin makroprosessoreiden ja niitä on tutkittu enemmän kuin mitään muuta siirrettävyyteen liittyvää välinettä.

3.7.1. Bootstrap

Nimi bootstrap, kengännauha, kuvaa hyvin toimenpidettä, jolla kääntäjä voidaan siirtää toiselle laitteelle. Tätä toimenpidettä tarvitaan silloin, kun kääntäjä on kirjoitettu ohjelmointikielellä, jolle ei kohdekoneessa vielä ole kääntäjää. Jos kääntäjä on kirjoitettu samalla ohjelmointikielellä, jota se kääntää, menetelmää on käytettävä. Jos kääntäjä on kirjoitettu toisella ohjelmointikielellä, vaatii se kääntäjän koodin siirrettävyyttä ja koodaukseen käytetyn kielen kääntäjää, jotta se voidaan kääntää kohdelaitteessa.

Half-bootstrap on toimenpiteenä seuraavanlainen. Kääntäjä on kirjoitettu jollain kielellä laitteessa A. Tällä kielellä tehdyt ohjelmat voivat kääntää laitteessa A laitteen objektikoodia. Kääntäjän koodia generoiva osa on sitten muutettu siten, että se tuottaa koneen B (kohdelaite) objektikoodia. Tämän B:n kääntäjän koodi käännetään sitten A:n kääntäjällä, jotta saadaan laitteelle A kääntäjä, joka tuottaa koneen B objektikoodia. Tätä uutta kääntäjää käytetään sitten kääntämään useimmiten ”itsensä”. Näin saadaan kohdelaitteen B objektikoodia oleva kääntäjä (Henderson, 1988). Useimmat siirrettävät kääntäjät siirretään tällä tavoin uusiin laitteistoihin. Samaa toimenpidettä voidaan käyttää myös makroprosessorien siirrossa toiseen laitteistoon.

Half-bootstrap vaatii laitteistojen välistä kommunikointia. Jos tämä on mahdotonta, on käytössä Full-bootstrap. Full-bootstrap:ssa laitteella B tehdään ensin yksinkertainen versio kääntäjästä laitteelle B, jota sitten käytetään tuottamaan kehittyneempi versio itsestään.

Full-bootstrap on monimutkainen ja vaikea tapa tehdä kääntäjän siirto toiselle laitteelle. Full-bootstrapin käyttö yksinkertaistuu, jos kääntäjä käyttää välikieltä. Välikielen tuottava osa kääntäjästä saadaan suoraan siirrettyä kohdelaitteelle, kunhan välikieltä tulkitseva osa on saatu tehtyä laitteistolle B. Tämän jälkeen kääntäjän kaikki ominaisuudet ja työkalut ovat käytössä, vaikkakin vajaatehoisesti tuottamaan tehokkaamman välikieltä tulkitsevan osan (Wallis, 1982).

Full-bootstrap voidaan toteuttaa myös tehdä useassa vaiheessa. Ensin asennetaan kohdelaitteistoon erittäin yksinkertainen makroprosessori SIMCMP, joka on yksinkertainen asentaa. SIMCMP on kirjoitettu FORTRAN:lla ja voidaan siirtää laitteeseen päivässä tai kahdessa. SIMCMP:n avulla asennetaan STAGE2, joka on koodattu kuvitteellisen laitteen assemblerilla FLUB:lla. FLUB:ssa on 28 konekäskyä ja kaksi pseudokäskyä. Nämä 30 makroa täytyy koodata jokaisella uudella laitteella. STAGE2:n asennus vaatii SIMCMP:n asennuksen, yksinkertaisen I/O-rajapinnan ja 30 yllämainittua makroa. Tähän muokkaukseen tarvittava työmäärä on arviolta kahdeksan tuntia. (Henderson, 1988)

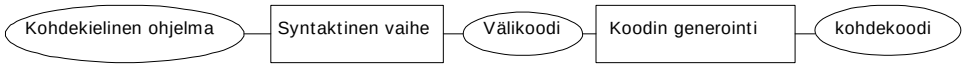
3.7.2. Välikieli

Kääntäjien suunnittelijat jakavat mielellään työn kahteen tai useampaan vaiheeseen, jotka tekevät oman osuutensa käännöksestä. Rajapinnat näiden vaiheiden välillä ovat välikoodeja. Tällä tavoin saadaan pieniä, koko kokonaisuutta helpommin hallittavia osia ja tällöin eri osia voidaan mahdollisesti käyttää hyväksi toisessa yhteydessä.

Välikieliä käytettäessä sen etu kääntäjän tai tulkin siirrossa on selkeä. Välikielen ollessa aina lähdekoodia yksinkertaisempaa, on sen asennus kohdekoneeseen yksinkertaisempaa kuin korkean tason kielen ja se saattaa olla jo asennettuna kohdelaitteistolle (Lecarme, 1989). Välikielen toteutukseen on useita ratkaisuvaihtoehtoja, jotka Wallis on jakanut alla olevan listan mukaisesti (Wallis, 1982).

- Välikieli voi olla korkean tason kielen kaltaista, jos kielessä on paljon epätavallisia piirteitä.
- Välikieli voi olla koodia, jossa on erotettu laitteistoriippuvaiset osat muusta koodista.
- Välikieli voi olla alemman tason ohjelmointikieltä, esimerkiksi assembleria.
- Välikieli voi olla kuvitteellisen laitteiston kone- tai ohjauskieltä.

Objektikoodin generointi on selkeästi laitteistoriippuvaista, kun taas laitteistosta riippumattomat osat kääntäjästä voidaan toteuttaa kaikilla laitteilla samalla tavalla. Siirrettävä kääntäjä on yleensä rakennettu alla olevan kuvan 1 mukaisesti. Kuvassa havainnollistetaan siirrettävän kääntäjän toteutusvaiheita. Syntaktinen osa kääntäjää on laitteistoriippumatonta ja koodin generointiosan ei tarvitse olla laitteistoriippumaton (Richards, 1979).



Kuva 1. Siirrettävän kääntäjän rakenne

Välikielen käsittelyyn voidaan käyttää joko kääntäjää tai tulkkia. Puhtaasti tulkkaavassa ratkaisussa käskyn suorittaminen voi vaatia melko monimutkaisen prosessin, jotta saadaan selville, mitkä operaatiot pitää suorittaa milläkin parametreilla. Useimmissa tapauksissa prosessi on identtinen kaikissa käskyn suorituskohdissa. Siitä johtuen, jos ohjelma on esimerkiksi kertautuvassa ohjelmarakenteessa, käsky tulkitaan useasti. Puhtaasti konekieliseksi käännetyssä koodissa jokainen käsky on käännetty kertaalleen (Ghezzi, 1982).

Yhden, useiden korkean tason ohjelmointikielien käyttämän välikielen idea tuli esiin IBM:n käyttäjäkerhon komiteassa, joka esitti käytettäväksi standardia, Universal Computer-Oriented Language, UNCOL. Jos kaikki korkean tason ohjelmointikielet käännettäisiin UNCOL:ksi, voitaisiin jokaisella koneella käyttää vain yhtä kääntäjää. UNCOL:in tavoitteet havaittiin liian vaikeiksi saavuttaa, ja projektia ei koskaan toteutettu loppuun asti. Idea kuitenkin on pohjana uusille projekteille, joista yksi on Janus (Wallis, 1982).

UNCOL:in kaaduttua mahdottomiin vaatimuksiinsa sekä kaikkien ohjelmointikielten tukemisesta, että kaikkien laitteistojen tukemisesta Waite alkoi kehittää Janus:ta, jonka tavoitteena on kehittää välikieli, joka toimisi kaikissa laitteistoarkkitehtuureissa. Janus määritteleekin kuvitteellisen laitteiston, johon on lisätty muutamia korkean tason ominaisuuksia. (Waite, 1975)

3.8. Muita työkaluja siirrettävyyden saavuttamiseksi

Siirrettävä tiedostojärjestelmä hävittää tiedostokäsittelyn erilaisuudet eri laitteistoilla. PDS on tällainen järjestelmä. PDS:n tarjoamia tiedostopalvelun primitiivejä ovat mm. mkfile, jolla luodaan tiedosto, ja openf avaa tiedoston käsittelyyn ja lukitsee sen muilta käyttäjiltä.

GENERATOR on FORTRAN -ohjelmien laitekohtaisia versioita tekevä generaattori. Ohjelmasta ylläpidetään yhtä versiota, josta luodaan GENERATOR:in avulla parametrointia hyväksi käyttäen siirrettävää koodia.

3.9. Siirrettävyys ja laatu

Ohjelmiston laatu on jotain,

- mitä kaikki haluavat
- minkä kaikki ymmärtävät
- mikä tulee luonnollisesti

- minkä puuttuessa syy on jonkun muun

Philip Crosby totesi näin kirjassaan jo vuonna 1979 (Crosby, 1979). Tosin hänen mielestään samat kohdat koskevat myös seksiä.

Tämä ajattelutapa on useissa ohjelmistotaloissakin muuttunut kansainvälisten laatuluokittelujen ja laatusertifikaattien myötä. Laatu koko toiminnassa ja laadun todentaminen ovat nyt huomattava osa ohjelmiston elinkaarta. Toisia laatuun vaikuttavia tekijöitä voidaan mitata, kuten ilmitulleet virheet jonkin ajan kuluessa, kun taas toisten tekijöiden mittaaminen, kuten ylläpidettävyyden, on suoranaisesti mahdotonta. Joka tapauksessa laatuun vaikuttavat tekijät tulee pystyä mittaamaan. Mittaaminen vaatii laadun osatekijöiden löytämistä.

Siirrettävyys on yksi ohjelmiston laadun osatekijöistä. Ohjelmiston laadun osatekijöiden vaikutus kohdistuu kolmeen ohjelmiston osaan, jotka ovat toiminnallisuus, muokattavuus ja yleiskäyttöisyys. Toiminnallisuuteen vaikuttavia tekijöitä ovat oikeellisuus, luotettavuus, tehokkuus, tiedon suojaus ja käytettävyys. Muokattavuus vaatii koodin ylläpidettävyyttä, joustavuutta ja testattavuutta. Yleiskäyttöisyys on koodin uudelleenkäytettävyyttä, mahdollisuutta liittyä muihin järjestelmiin ja siirrettävyyttä (Pressman, 1990).

Ohjelmiston laatuvaatimukset pitää ottaa huomioon elinkaaren alkuvaiheessa. Laatu tulee esiin usein elinkaaren loppuvaiheessa ja laadun puuttumisen hinta on usein suurempi kuin laadun tekemisen hinta. Siirrettävyys tulee ottaa huomioon ohjelmiston suunnittelussa ja koodauksen aikana. Vaikutukset näkyvät sitten ohjelmiston yleiskäyttöisyydessä elinkaaren lopussa. Hinta yleiskäyttöisyyden saavuttamiselle muokkaamalla ohjelmistoa elinkaarensa lopussa on huomattava verrattuna hintaan, joka on maksettava laadusta ohjelmiston suunnittelu- ja koodausvaiheessa (Schulmeyer, 1990).

Pressman määrittelee kirjassaan (Pressman, 1990) siirrettävyyden vaikutukset laadun mittareihin seuraavasti. Siirrettävyys vaikuttaa kahdestakymmenestä kahdesta laadun mittarista viiteen, jotka ovat yleisyys, laitteistoriippumattomuus, modulaarisuus, itsedokumentoivuus ja järjestelmäriippumattomuus.

Laadun määrän mittaaminen on vaikeaa, mutta Halstead (Halstead, 1977) on tehnyt matemaattisia malleja laadun määrän mittaamiseen. Halsteadin mukaan ohjelman ja ohjelmiston laatu voidaan määritellä ohjelmiston toteutuksen ja sen teoreettisen tehokkaimman toteutuksen suhteena. Ohjelmiston monimutkaisuus lasketaan ohjelmiston muuttujien ja operaatioiden funktiona. Laatu saadaan laskentakaavoista arvioimalla ohjelman koodin pituutta, ohjelman monimutkaisuutta ja käyttämällä näiden lisäksi ohjelmistokielelle laskettua vakiota. Laskennallinen paras toteutus saadaan selville ja tätä verrataan toteutettuun sovellukseen. Ongelmana tässä kaavassa on ohjelmistokielelle laskettu vakio, jonka on havaittu olevan riippuvainen myös ohjelmoijasta.

McCall rakentaa hierarkkisen mallin (Curtis, 1980), joka kuvaa ohjelmiston laadun ominaisuuksia. Mallissa ylimpänä tasona on ensisijainen käyttö (primary uses).

Ensisijainen käyttö voi olla ohjelmiston käyttö sellaisenaan, ohjelmiston käyttö muokattavana tai ohjelmiston yleinen käyttö. Toinen taso on välitason rakenne, joihin kuuluu myös siirrettävyys. Siirrettävyys kuuluu ohjelmiston yleiseen käyttöön. Muita välitason rakenteita ovat luotettavuus, tehokkuus, käytettävyys, testattavuus, ymmärrettävyys ja muutettavuus. Kahdestatoista alemman tason rakenteesta siirrettävyys vaikuttaa kahteen, laitteistoriippumattomuuteen ja ohjelmiston valmiuteen. Nämä alemman tason rakenteet vaikuttavat sitten kaikki laadun mittareihin. Laatua ei koskaan voida mitata tarkasti, mutta sen vaikutusta laadun osatekijöihin voidaan arvioida. Ohjelmiston suunnittelun, toteutuksen, ominaisuuksien ja rakenteen vaikutukset näihin laadun osatekijöihin täytyy tarkastella kriittisesti, jotta voidaan osoittaa selkeitä syy-seuraussuhteita.

Siirrettävyyden vaikutus järjestelmän ominaisuuksiin tulee esiin erityisesti pitkän elinkaaren kohdalla. Järjestelmän, erityisesti paljon laskentaa, yleisiä lainalaisuuksia tai lakisääteisiä ominaisuuksia sisältävän ohjelmiston pitkä elinkaari on yksi tärkeimmistä ominaisuuksista. Vaikutus ei ole laatutekijöiden kohdallakaan yksisuuntaista, vaan pitkän elinkaaren omaava ohjelmisto tai sen osa vaikuttaa myös pitkän elinkaaren muihin osatekijöihin, kuten ylläpidettävyyteen ja muokattavuuteen. Pitkä elinkaari tuo myös mukanaan ohjelmiston monipuolisen käytön ja tämä vaikuttaa ohjelmiston oikeellisuuteen. Pitkän käyttöiän aikana mahdolliset virheet tulevat esiin, jolloin ne pystytään korjaamaan.

Siirrettäväksi suunnitellun ohjelmiston tulee olla myös modulaarinen, jotta siirrettävyyden mahdolliset ongelmat pystytään eristämään ja ratkaisemaan. Ohjelmiston datan siirrettävyydestä tulee myös pohja ohjelmiston liitettävyydelle muihin ohjelmiin. Datan oikea muoto antaa suuntaviivat ohjelmiston liittymärajapintojen rakentamiseen.

Haitallinen vaikutus siirrettävyydestä saattaa olla ohjelmiston tehokkuudelle. Käyttöjärjestelmien ja laitteiden ominaisuuksia ei mahdollisesti voida käyttää optimaalisesti. Tällainen ominaisuus on esimerkiksi lajittelu, joka voidaan usein optimoida laitteelle ja tulee joskus laitteen varusohjelmiston mukana. Siirrettävyys luo myös rasitteita ohjelmiston suunnittelussa ja koodauksessa muuttujien ja ohjelmointikielten erityispiirteiden käyttöön. Usein kaikkein hankalimmat toteutuksen ongelmat on ratkaistu ohjelmointikielissä tai kehittimissä jollain järjestelmän tai ohjelmointikielen erityispiirteellä, joka ei ole standardi. Toimintojen suunnittelu ja toteuttaminen ilman näitä erityisiä apukeinoja vaatii kurinalaisuutta ja selkeää modulaarisuutta siinä tapauksessa, että epästandardeja tapoja käytetään.

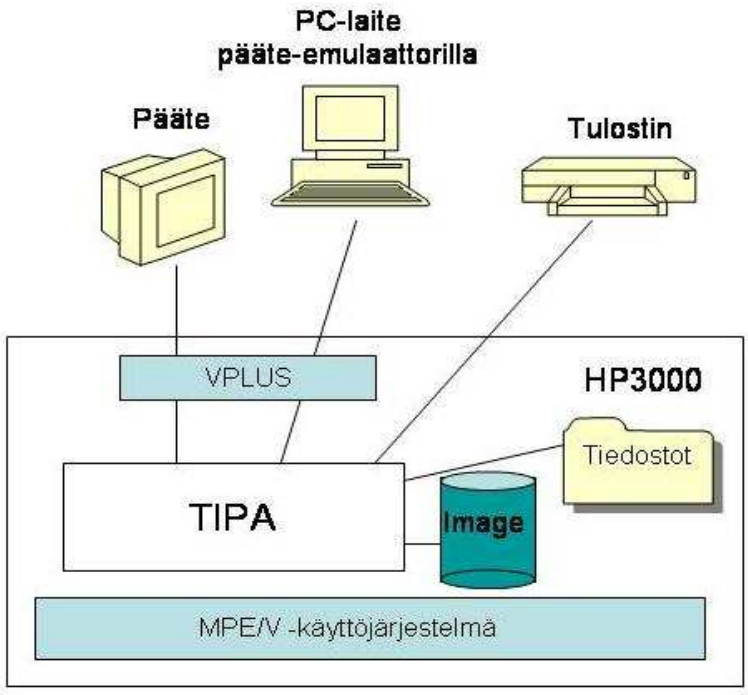
4. SIIRRETTÄVÄ COBOL-OHJELMISTO

4.1. TIPA-ohjelmisto

TIPA on Tietonauha-yhtiön henkilöstöhallinnon tietojärjestelmä, jonka kehitys on aloitettu 1980-luvun puolivälin jälkeen. Järjestelmä on joustava päätekäyttöinen ohjelmisto, jonka avulla voidaan hoitaa keskeiset henkilöstöhallinnon tehtävät:

- Henkilöstösuunnittelu.
- Henkilöstön hankinta.
- Henkilöstön kehittäminen.
- Palkanlaskenta ja tilastointi.
- Henkilöstöpalvelut.
- Palkkabudjetointi ja seuranta.

Järjestelmän toteutettiin HP3000-laitteistoon, jossa käytettiin laitteistoon ja käyttöjärjestelmään tiukasti sidottuja ohjelmointivälineitä. Kuvassa 2 on TIPAn alkuperäinen arkkitehtuuri. Järjestelmää käytetään joko päätteiltä tai emulaattorin avulla PC-laitteilta. Kaikki HP3000-laitteella TIPAan liittyvät varusohjelmistot ovat sidottuja MPE/V-käyttöjärjestelmään ja kyseiseen laitteeseen.



Kuva 2. TIPAn arkkitehtuuri

TIPAa voidaan soveltaa kaikkiin työehtosopimuksiin, koska työehtosopimus- ja toimialakohtaiset sekä yrityskohtaiset käsittelysäännöt ovat pääsääntöisesti ohjaustiedoissa erillään ohjelmista. Yleisimpiin työehtosopimuksiin on valmiit ohjaustiedot: mm. metalli, mekaaninen ja kemiallinen puunjalostus, kemia, graafinen, elintarvike, ahtaus- ja kuljetus, voimateollisuus, rakennusteollisuus, teleliikenne, puhtaanapito ja pesulaliiketoiminta.

TIPA on aluksi toteutettu keskitetyksi ohjelmistoratkaisuksi, jolla voidaan yhdessä laiteympäristössä hallita kaikki palkka- ja henkilöstöhallinnon toiminnot. Siirrettävyyden toteutuksen jälkeen sama ohjelmisto on käytettävissä useissa eri käyttöjärjestelmissä ja laitteissa antaa mahdollisuuden toteuttaa yrityskohtaiset ratkaisut.

Ohjelmia TIPAssa on yli 1200 ja COBOL-koodia yli 300 000 riviä. TIPAa on kehitetty yli 40 henkilötyövuoden työpanoksella, ja suuri osa työstä on mennyt järjestelmän lakisääteisten ominaisuuksien toteuttamiseen ja järjestelmän sovittamiseen erilaisten työehtosopimusten ja toimintatapojen mukaiseksi.

4.2. Tietovarastot

Ohjelmiston perustan muodostaa erityyppisten toimialojen tarpeita varten suunniteltu tietokantarakenne, jossa tiedot on tallennettu Image-tietokantaan. Lisäksi ohjelmiston dataa on tallennettu indeksoituihin tiedostoihin sekä parametritiedostoihin niitä ohjelmiston osia varten, jotka eivät pysty lukemaan tietokantaa ja hakemaan sieltä tarvitsemaansa tietoa.
Taulukoissa 1-4 on listattuna kantaan tallennetut tiedot.

Henkilöiden perustietokannat	
Tieto	**Sisältö**
Henkilön perustiedot	Henkilön tiedot, mm. Nimi, henkilötunnus
Palkkatiedot	Palkkatieto ja palkkahistoria
Työkokemustiedot	Vanhat kokemustiedot, nykyiset tiedot ja poissaolot
Koulutus	Taidot ja kurssit

Taulukko 1, Henkilöiden perustietokannat

Ohjaustietojen perustietokannat	
Tieto	**Sisältö**
Palkkalajisto	Tiedot palkkalajeista
Kalenteri	Työaikakalenterit eri sopimusten mukaan
Työtuntijärjestelmät	TES-sopimusten perustiedot
Ohjaustiedot	Parametrit mm. Käytettäville TES-tulkinnoille

Taulukko 2, Ohjaustietojen perustietokannat

Kertymätietokannat	
Tieto	**Sisältö**
Lomatiedot	Henkilöittäin, palkkakausittain sekä palkkalajeittain tunnit ja markat
Edellisten seurantakausien tiedot	Henkilöittäin, palkkakausittain sekä palkkalajeittain tunnit ja markat

Taulukko 3, Kertymätietokannat

Tapahtumatietokannat	
Tieto	**Sisältö**
Henkilön perustiedot	Henkilön tiedot, mm. Nimi, henkilötunnus
Palkkatiedot	Palkkatieto ja palkkahistoria
Työkokemustiedot	Vanhat kokemustiedot, nykyiset tiedot ja poissaolot
Koulutus	Taidot ja kurssit

Taulukko 4, Tapahtumatietokannat

4.3. Ohjelmiston toiminnallisuuksia

Henkilöstösuunnittelu. Järjestelmän avulla on käytettävissä ajan tasalla olevat tiedot yrityksen palkkapolitiikan kehittämiseksi ja resurssien suuntaamiseksi haluttuun suuntaan. Urakehityksen tarvitsemat perustiedot koulutuksesta, taidoista ja työkokemuksesta ovat järjestelmän perusominaisuuksia. Valmiit raportit henkilöstön vahvuudesta, vaihtuvuudesta sekä ikärakenteesta antavat pohjan realistiselle suunnittelulle. Palkkakehitystä voidaan seurata henkilöittäin sekä organisaation mukaisesti halutulla aikavälillä.

Henkilöstön hankinta. Henkilöiden perustietoja voidaan tallettaa työsuhteessa olevien lisäksi myös työnhakijoista paikanhakulomakkeessa annettujen tietojen perusteella, jolloin tietoja voidaan käyttää henkilöstön hankintaa ja työhönottoa varten.

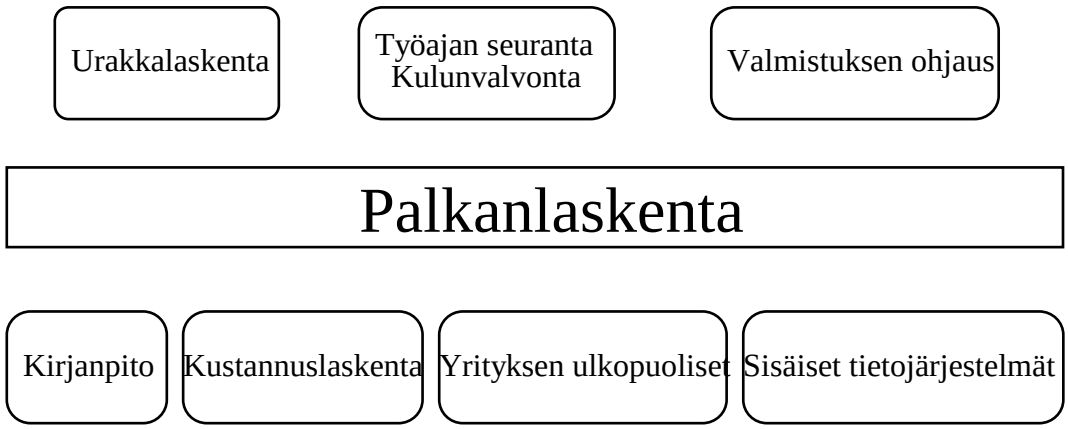
Henkilöstön kehittäminen. Henkilöstön kehittämisessä ovat ajan tasalla olevia tärkeitä tietoja mm. tiedot peruskoulutuksesta, tutkinnoista, kielitaidosta ja kursseista. Kurssiseurantaa voidaan hyödyntää täydennyskoulutuksen suunnittelussa ja laatutoiminnassa.

Palkanlaskenta. Palkanlaskennan keskeisiä piirteitä ovat:

1. Tunti-, urakka- ja kuukausipalkkojen laskenta sekä myös työntekijöiden kausipalkkojen laskenta. Tapahtumien käsittelyssä ovat useat erilaiset valmiit rutiinit käytettävissä, kuten myös kalenteriin ja työ-tuntijärjestelmään tukeutuvat automaattiset tapahtumien muodostamisrutiinit. Näitä elementtejä hyväksi käyttäen on myös toteutettu poikkeamaperusteinen tapahtumakäsittely. Mikäli tapahtumissa on myös tekopäivä, kerätään automaattisesti lomaoikeustiedot sekä pekkaspäiviin yms. työajan lyhentämisen seurantaan liittyvät tekijät.

2. Palkkojen välitys konekielisesti pankkeihin, tallennettuna siirrettäväksi tai esim. KOKOPANKKI-tuotetta hyväksikäyttäen linjasiirtona.
3. Lopputilien, loma yms. palkkojen maksatusajot haluttuna ajankohtana. Lopputilien laskennassa automaattinen lomakorvausten käsittely.
4. Lakisääteinen ja TES-pohjainen palkkatilastointi.
5. Keskituntiansioiden laskenta halutulla tavalla. Lomakeskiansion lisäksi on mahdollista hallita neljää erilaista keskituntiansiota/työehtosopimus.
6. Lomapalkkojen ja lomaltapaluurahojen laskenta. Lomapalkat voidaan haluttaessa maksattaa automaattisesti henkilötietoihin talletettuihin lomanpitoaikoihin perustuen.
7. Ylitöiden hälytysraportointi. Perinteisen ylityötilastoinnin lisäksi tuotteessa on monipuolinen ajankäytön raportointi. Käyttäjä voi palkkalajien perusteella seurata samalla raportilla mm. työaikaa, ylitöitä sekä valittuja poissaoloja.
8. Poissaolojen seuranta kalenterikuukausittain.
9. Jäsenmaksujen laskenta ja tilastointi.
10. Yleiskorotusten laskenta usealla eri tavalla.
11. Palkkatilastointi työnantajaliitoille, vakuutus- ja eläke-laitoksille ja veroviranomaisille. Liittymät mg-nauhalla tai linjasiirtona sen mukaan, miten kyseinen vastaanottaja edellyttää.
12. Liittymät muihin ohjelmistoihin.

TIPAn palkanlaskentasovelluksessa on liittymät kuvassa 3 näkyviin ohjelmistoihin. Liittymät ovat eräsiirtoja ja tiedot siirretään tiedostoina suurimmaksi osaksi saman laitteen sisällä tai tallennusmedian avulla saman yrityksen toiseen laitteeseen.

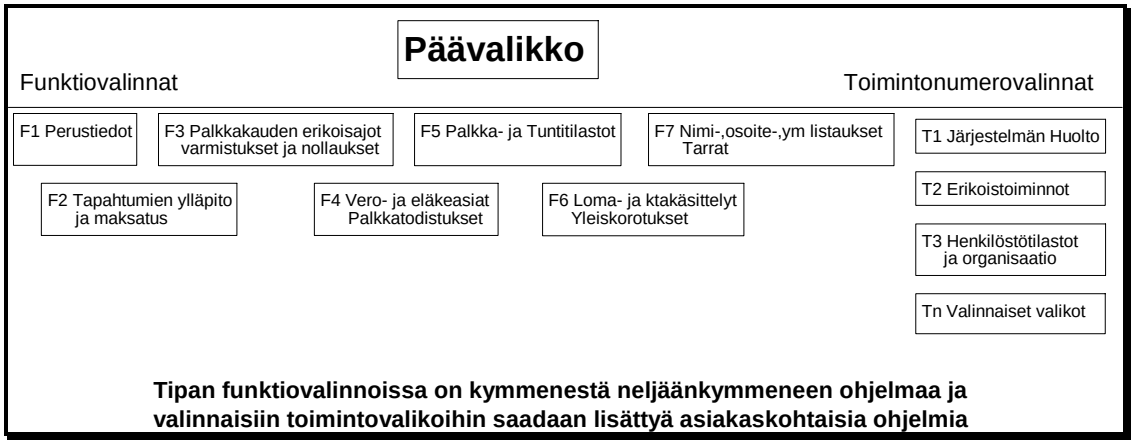


Kuva 3. Palkanlaskennan liittymät

4.4. TIPAn käyttö

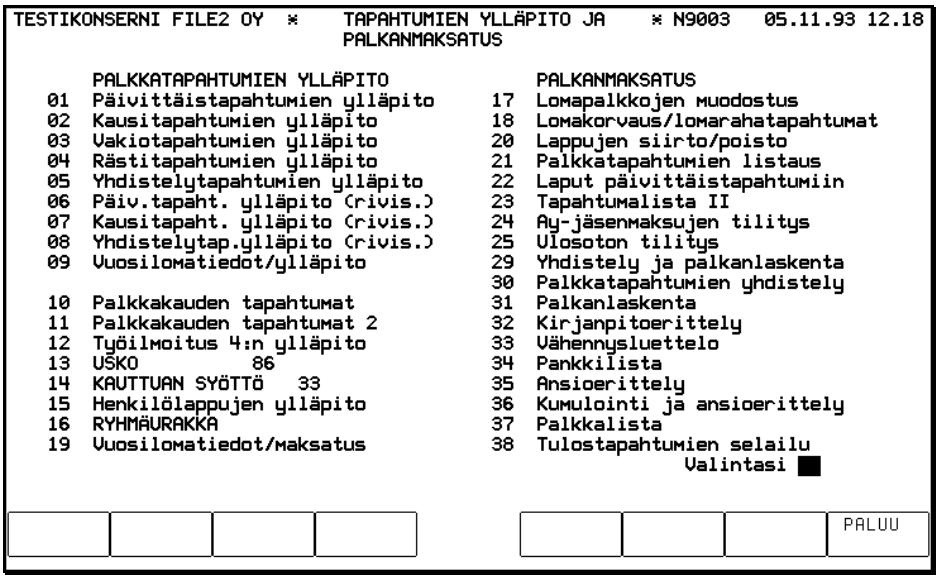
TIPA on valikkopohjainen palkanlaskentajärjestelmä, jonka valikoissa liikutaan funktionäppäimiä painamalla. Ohjelmat ovat valikoissa aiheittain jaoteltuina. Tietojen päivitykset ja ajojen käynnistykset tapahtuvat näppäimistöltä ohjelmiston näyttöjen kenttiin kirjoittamalla ja funktionäppäimiä painamalla. Ohjelmistoon kirjoittaudutaan

avausnäyttöjen kautta. Avausnäytöillä kysytään ohjelmiston käyttäjätunnusta ja salasanaa. Avausnäytön jälkeen avautuu TIPAn päävalikko (Kuva 4), josta päästään funktionäppäimillä valitsemalla eri toimintovalikoihin.



Kuva 4. TIPAn valikot

TIPAn valikkopuu on kolmitasoinen. Päävalikosta voidaan valita funktionäppäimillä seitsemän eri valintaa (F1 - F7), jotka ovat kiinteästi määrättyjä ohjelmistossa ja numerovalinnoilla aina kolmeentoista toimintovalintaan asti ohjelmallisesti määrättäviä valintoja. Funktionäppäimien ja toimintonumeroiden käynnistämiltä näytöiltä löytyvät TIPAn ohjelmat, joita on esim. funktiovalinnan 2 alla 35 kappaletta. Kuvassa 5 näkyvät päävalikon valinnasta F2, Tapahtumien ylläpito ja maksatus, avautuva toimintovalikko.



Kuva 5. Esimerkkivalikko TIPAsta, Tapahtumien ylläpito

Ohjelmiston näytöt ovat merkkipohjaisia ja ne luetaan enter-näppäimen painalluksella, jonka jälkeen näytön tietojen oikeellisuus tarkastetaan ja näytön käsittely siirtää tiedot ohjelmille.

4.5. Ohjelmien jaottelu

TIPAn ohjelmat voidaan jakaa kolmeen ryhmään. Taulukossa 5 on listattuna esimerkkejä erityyppisistä TIPAn ohjelmista.

Vuorovaikutteiset pääteohjelmat	Laskentaohjelmat	Tulostusohjelmat
henkilötietojen ylläpito	palkanlaskenta-ajo	henkilötietojen muutoslistaus
palkkatapahtumien ylläpito	keskituntiansioiden laskenta	tuntitilasto
ohjaustietojen ylläpito	lomapalkkojen laskenta	Työvoiman käytön tiedustelu
palkkalajitietojen ylläpito	yleiskorotusten laskenta	Lomatietojen listaus

Taulukko 5, TIPAn ohjelmatyypit ja esimerkkiohjelmat

Vuorovaikutteiset pääteohjelmat käsittelevät tietokantoja reaaliaikaisesti. Pääteohjelmilla suoritetaan palkkajärjestelmien tietojen ylläpito. Tietojen päivitys tapahtuu näppäimistöltä ja näytölle syötetyt tiedot luetaan järjestelmään enter-näppäimen painalluksella. Syötettyjen tietojen oikeellisuus tarkastetaan ja mahdolliset virheilmoitukset ja ohjeet näytetään käyttäjille.

Laskentaohjelmat suorittavat laskentaa tietokannan tiedoista. Laskentaohjelmat käynnistetään näytöiltä. Laskentaohjelmat hakevat tietoja tietokannoista ja käyttävät joissain tapauksissa myös indeksoituja tiedostoja ohjaustietoja hakiessaan. Laskennan tuloksena syntyneet tiedot talletetaan tietokantaan.

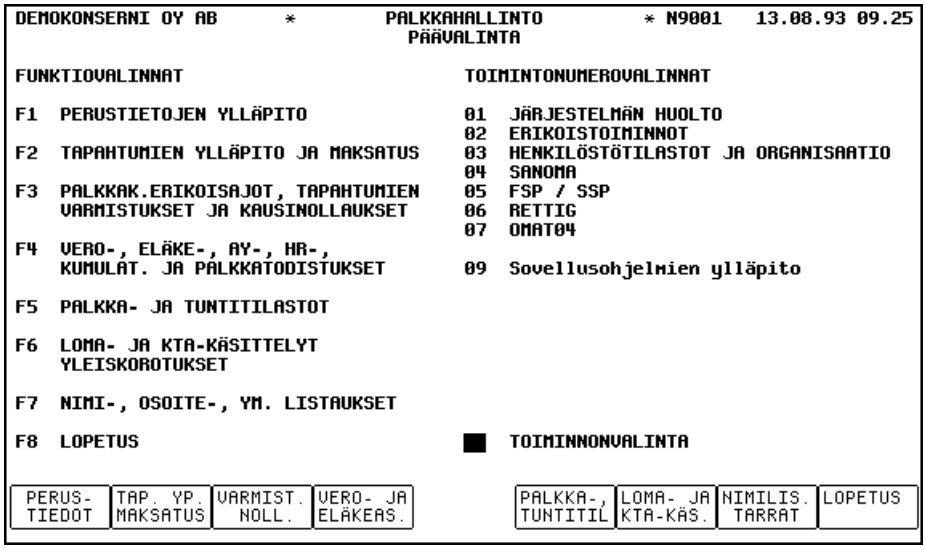
Tulostusohjelmilla palkkajärjestelmä tulostaa erilaisia raportteja palkansaajille, pankeille, kirjanpitoa varten, yrityksen ja osastojen johtotehtävissä toimiville, muille yrityksen sisäisille kohteille, ammattiliitoille, veroviranomaisille, eläkelaitoksille, työnantajaliitoille ja vakuutuslaitoksille. Raportteja löytyy järjestelmästä yli 100 kappaletta ja ne on jaoteltu järjestelmässä käyttäjäryhmittäin.

Näiden ohjelmien lisäksi tuotteessa on raporttigeneraattori, jolla käyttäjä itse pystyy tuottamaan haluamansa sisältöisen raportin. Järjestelmän tiedot on suojattu ja tietosuojavaltuuksien puitteissa henkilö saa vain ne tiedot, joihin hänellä on valtuudet.

4.6. TIPAn valikot

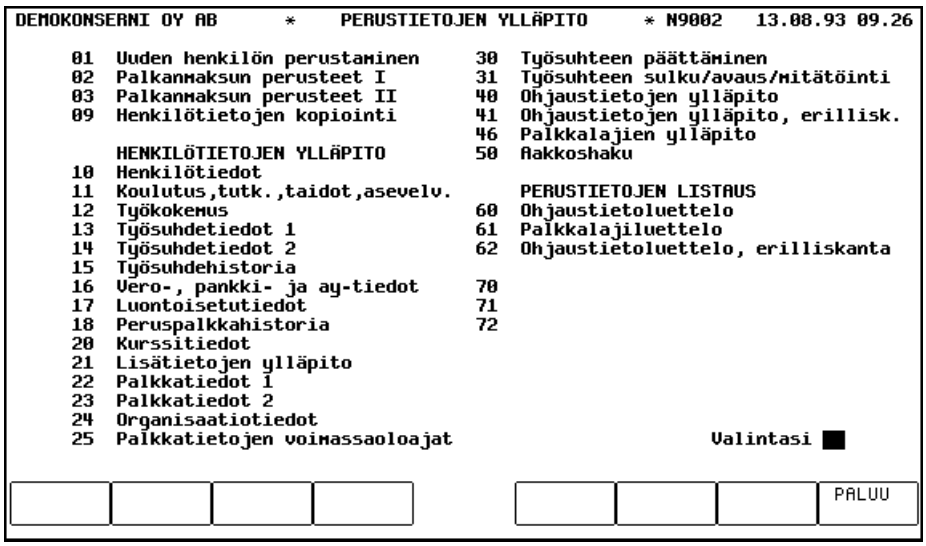
Perusohjelmiston rakenne on ns. yleispätevä eli ajateltuna puhtaasti palkanlaskennan näkökulmasta. Varsinkin suuremmissa yrityksissä on työnjako eriytynyt esim. palkkakonttorin ja henkilötoimiston välillä. Tällöin perusvalinnat muokataan ko. asiakkaan työnjakoja vastaaviksi. Tapahtumatietojen ylläpito on myös rutiini, joka yleensä muokataan asiakkaan tarpeita vastaavaksi. Kuvissa 6-9 ovat esimerkkikuvaukset TIPAn standardimenuista. Kuvassa 6 on eräs malli TIPAn

päävalikosta. Päävalikkoon tuodaan usein eri käyttäjäryhmille yleisimmin käytetyt ohjelmat ja valikot.



Kuva 6. TIPAn päävalikko

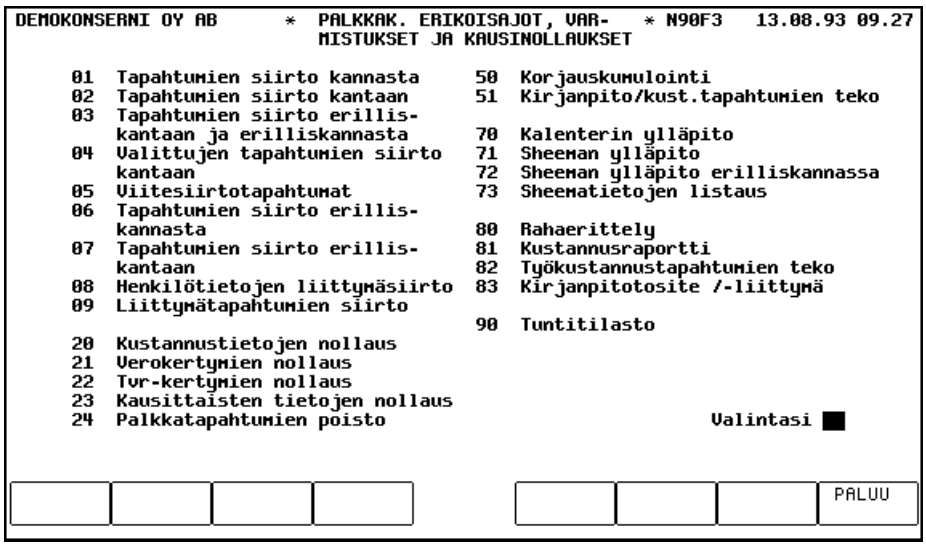
Vuorovaikutteisia pääteohjelmia on sekä perustietojen ylläpidon valikossa, joka on kuvassa 7, sekä tapahtumien ylläpidon valikossa. Perustietojen ylläpidon valikon toiminnallisuudet on nimetty selkeästi ja ne on ryhmitelty käytettävyyden parantamiseksi.



Kuva 7. Perustietojen ylläpitovalikko

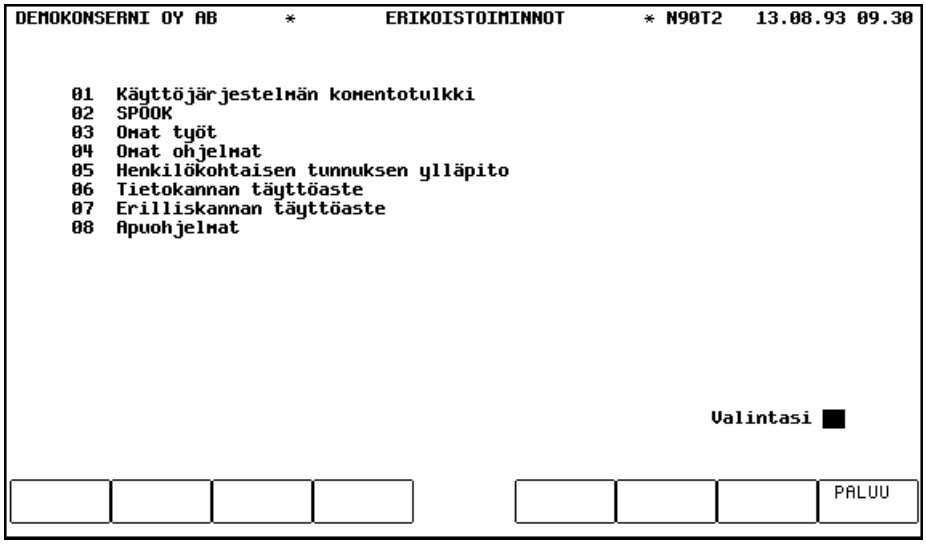
Laskentaohjelmien käynnistykset tapahtuvat mm. kuvan 8 valikosta palkkakauden erikoisajot, varmistukset ja nollaukset. Laskentaohjelmat muuttavat tietoja kannassa ja listaavat laskennan lopuksi yhteenvetotiedot tehdyistä toimenpiteistä. Suurimmalle osalle laskentaohjelmia ei löydy erillisiä ohjelmia toimintojen perumiseen, mutta

virhetilanteessa ohjelmat pystytään ajamaan uudestaan ja korjaamaan mahdolliset virheet.



Kuva 8. Laskentaohjelmia

Pääkäyttäjille on järjestelmässä omia toiminnallisuuksia, joilla he voivat hallinnoida järjestelmää ja seurata järjestelmän käyttöä. Kuvassa 9 on pääkäyttäjän valikko, jolla pääkäyttäjä pääsee käyttöjärjestelmätasolle ja voi ajaa apuohjelmia. Erityisesti tietokantojen täyttöasteiden seuranta on tarpeellista, koska tietokannat eivät kasva dynaamisesti, vaan tietokantojen varaamaa levytilaa pitää kasvattaa tarpeen mukaan.



Kuva 9. Pääkäyttäjän erikoistoiminnot

5. KONVERTOINTI

5.1. Yleistä

HP-COBOL -ohjelman muokkaaminen siirrettäväksi sai alkunsa vuonna 1990, kun Tietonauhan palkka- ja henkilöstöhallinnon ohjelmiston, TIPAn kehittäminen ja laitepohjan laajentaminen tulivat ajankohtaisiksi. HP3000 -laitteistot olivat jäämässä ns. avoimen ympäristön järjestelmien, pääasiassa Unix-laitteiden jalkoihin yritysten laitehankinnoissa. Yrityksille muotoiltiin nyt standardit tietotekniikan kehittämisessä ja tämä standardi tarkoitti suurimmassa osassa yrityksiä Unix -käyttöjärjestelmän käyttöönottoa.

HP3000 -laitteiden sopivuus ja toimivuus henkilöstöhallinnon kaltaisten järjestelmien käytössä eivät olleet vieläkään kyseenalaisia. Tietokantana HP3000:lla käytetty Image on omiaan suurten massojen käsittelyyn, jota jopa neljännesmiljoona palkkatapahtumaa kuukaudessa läpikäyvä palkkahallinnon järjestelmä edustaa. Erätyöt ja tulosteiden helppo hallinta tukevat mainiosti TIPAn kaltaisen ohjelmiston toimintaa. Käyttöjärjestelmään kiinteästi liittyvä näytönkäsittelyn työkalu VPLUS, sekä muut käyttöjärjestelmään integroidut työkalut tekevät HP3000 -ympäristöstä, muiden suljettujen järjestelmien tavoin, erinomaisia laitepohjia suurten ohjelmistojen rakentamiseen.

Aika ja ennen kaikkea raha näyttävät kuitenkin ajavan ohi suljetuista järjestelmistä. Avointen järjestelmien laaja ohjelmisto- ja työkalutarjonta tekevät yhdessä laitteen halvan hinnan kanssa näistä järjestelmistä houkuttelevan vaihtoehdon yrityksen tulevaa tietojenkäsittelyn laitekantaa valittaessa.

Unix on käyttöjärjestelmä, joka on usein suosituin valinta tietojärjestelmiään kehittävässä yrityksessä. Vaikkakin Unix ja C-kieli, jolla Unix itsessään on toteutettu, ovat avointen järjestelmien tärkeimpinä osina tarkoitettu takaamaan yhtenäinen pohja ohjelmistoille ja muille työkaluille, on niiden standardointi ja siirrettävyys vielä alkutaipaleellaan. Vaikeakäyttöisyys ja kehittymättömyys ovat saaneet aikaan sen, että Unixista on jokaisella merkittävällä laitevalmistajalla oma versionsa ja C-kielissä on useita toisistaan poikkeavia ominaisuuksia.

Käyttöjärjestelmän ja siihen integroitujen tuotteiden vähäinen tuki ohjelmistoille vaatii niiltä enemmän. Tulosteiden ja erätöiden hallinta täytyy hoitaa ohjelmallisesti. Näiden käyttöjärjestelmään liittyvien ohjelmiston osien sisällyttäminen ohjelmistoon eristämällä käyttöjärjestelmän kutsut esimerkiksi ohjelmiston ylläpitämiin ohjaustietoihin tekee ohjelmiston mahdollisimman vähin muutoksin siirrettäväksi.

TIPAn siirtotyössä päätettiin säilyttää COBOL-koodin kirjoittamiseen käytetty työ. Vaihtoehtoina olivat ohjelmiston kirjoittaminen uudelleen esimerkiksi C-kielellä tai

jollain neljännen sukupolven ohjelmankehittimellä kuten PROGRESS. TIPAan kuuluvien on-line ohjelmien ja eräohjelmien yli 120 000 riviä koodia olisivat kuitenkin vaatineet senkaltaisen työpanoksen, johon ei ollut varaa, koska samaan aikaan HP3000:lla olevaa ohjelmistoa ylläpidettiin ja kehitettiin. Vaihtoehdoksi jäi muokata COBOL-koodi jollekin siirrettävälle COBOL-kielelle. Näistä valittiin AcuCOBOL-85. AcuCOBOL on Ansi 85 -standardin mukainen COBOL -kääntäjä, jonka siirrettävä välikieli on ajettavissa AcuCOBOL -runtime:lla. AcuCOBOLin ajoversiota on useille käyttöjärjestelmille ja yli 600 eri tietokoneelle.

HP3000:lla käytetyt näytönkäsittely (VPLUS) ja kannankäsittely (IMAGE) olivat, yhdessä erätöiden ja tulostuksen hallinnan kanssa, ne osat, joiden muuttamisessa siirrettäväksi oli suurin työ. Tavoitteena oli luoda järjestely, jolla mahdollisimman automaattisesti voitaisiin siirtää HP3000:lla ylläpidettävä ohjelma muuttuneilta osiltaan muita laitteita käyttäville asiakkaille.

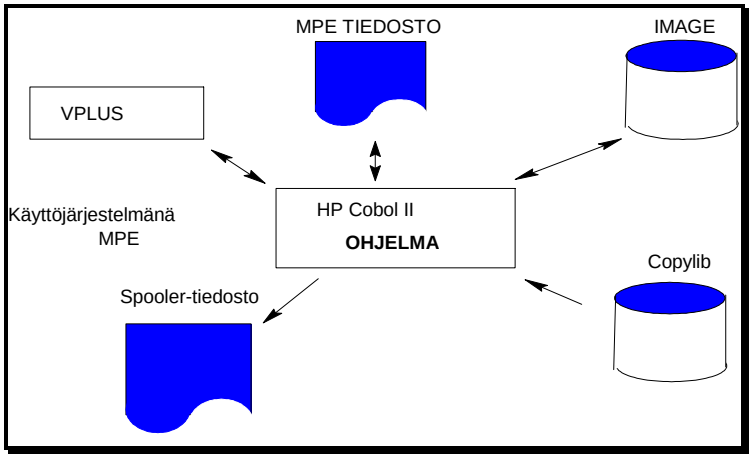
Siirtotyön toteutuksen laitteeksi valittiin PC Windows-käyttöjärjestelmällä, joka edullisuudellaan ja työkalujensa monipuolisuudella oli selvästi muita vaihtoehtoisia laiteympäristöjä parempi. Esimerkiksi ohjelmistoa voitiin PC:ltä ajaa yhtä aikaa sekä HP3000:lla, PC:llä että HP9000 -Unix-laitteessa ja testaus oli helppo suorittaa.

5.2. Konversion laitteistot

5.2.1. Lähdejärjestelmän laitteet ja työkalut

Ohjelmisto on pääosin kehitetty 80-luvun lopulla Hewlett-Packardin HP3000-laitteistolla, joka on erityisesti tapahtumankäsittelyyn tehty tietokone. Siirtyminen muihin laiteympäristöihin on toteutettu vaihtamalla HP3000:n tiukasti käyttöjärjestelmäsidonnaiset ohjelmistokehityksen työkalut useissa laiteympäristöissä toimiviin ohjelmistotuotteisiin.

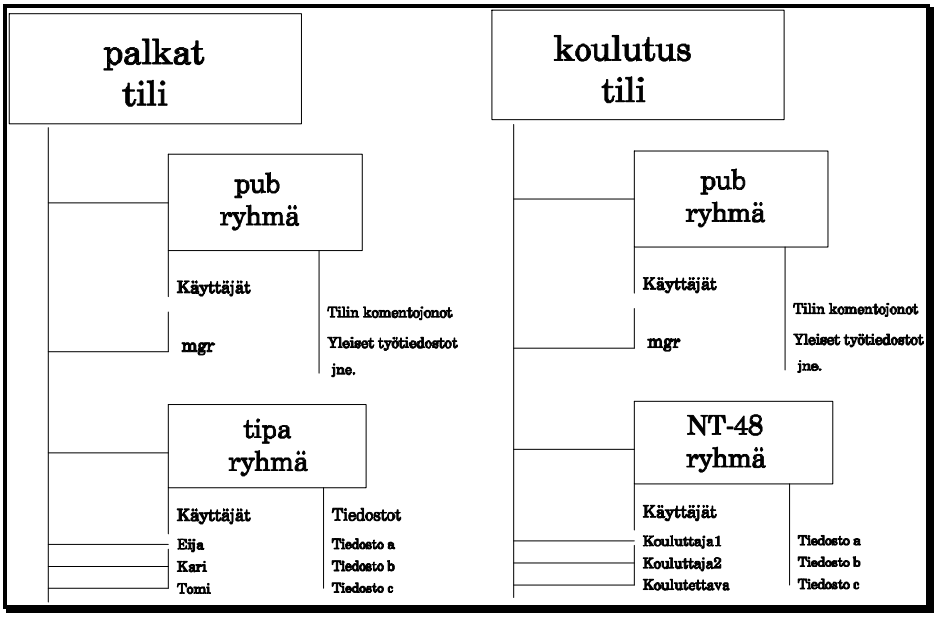
TIPA on HP3000-koneella kehitetty ja sen kehitykseen on käytetty MPE-käyttöjärjestelmään tehtyjä ohjelmatuotteita, joilla on saatu aikaan kattava ohjelmointiympäristö. Kuvassa 10 näkyvät ohjelmien käyttämät resurssit.



Kuva 10. Ohjelman ympäristö HP3000:lla

Ohjelmat on kirjoitettu HP COBOL II- ohjelmointikielellä, joka noudattelee COBOL-85-standardia HP-lisäyksin. MPE on erityisesti tapahtumankäsittelyyn soveltuva käyttöjärjestelmä, jolle ovat tyypillisiä tunnuksia moniajo ja töiden ajastaminen sekä monipuoliset tiedostonkäsittelymahdollisuudet. MPE/XL mahdollistaa 64-bittisen osoiteavaruuden ja muistiin kuvattujen (mapped) tiedostojen käytön. Toiminnan tehokkuus perustuu laajan osoiteavaruuden käytön lisäksi suureen keskusmuistin määrään ja laajan levytilan käyttöön. MPE/XL-käyttöjärjestelmän muistiin kuvattujen tiedostojen käyttö on kehitetty aikaisemmissa MPE/V-käyttöjärjestelmissä käytetystä levyvälimuistiominaisuudesta (disc cache). Muistissa tallessa olevat tiedot, ohjelmat ja data saadaan nopeasti käyttöön ja hitaat levy I/O-tapahtumat vähenevät. Tiedostojen muistiin kuvaus perustuu MPE/XL:n sivutettuun virtuaalimuistiin. Avoimeen tiedostoon ja sen sisältöön voidaan viitata virtuaalimuistin avulla. Jokaisella tavulla ja avatulla tiedostolla on yksikäsitteinen virtuaaliosoite.

Kuvassa 11 tilirakenteen osalta esitelty MPE-käyttöjärjestelmän tili- ja tiedostojärjestelmä on suunniteltu erityisesti teollisuuden ja kaupan sovelluksia silmälläpitäen. Koneen levytilaa jaetaan systeemissä tileihin (account), jotka jakaantuvat ryhmiin (group). Tiedostot sijaitsevat ryhmien alla ja kuuluvat siis jonkin tilin ryhmään.



Kuva 11. HP3000-tietokoneen tilirakenne

Järjestelmät on suojattu tehokkaasti, sillä jokaisella tilillä, ryhmällä ja tiedostolla voi olla oma salasanansa ja jokaisella käyttäjällä omat valtuutensa erityyppisten tiedostojen käsittelyyn. Käyttäjätunnukset ovat tilikohtaisia. Käyttäjät kirjoittautuvat oman tilinsä kotiryhmään.

Kopiokirjasto on Copylib-nimisen työkalun avulla tehty joukko yleisiä ohjelmarutiineja ja tiedostomäärittelyjä. Copylib:ssa kirjaston osat ovat yhtenä tiedostona, jota voidaan editoida myös HP3000:n Qedit-editorilla.

Tiedostot ovat MPE-tiedostoja tai erilaisia ominaisuuksia omaavia erikoistiedostoja. MPE-käyttöjärjestelmässä on useita erilaisia tiedostomuotoja, joita on käytetty hyväksi ohjelmallisiin tarkoituksiin. Tiedostojen käsittelyyn on HP COBOL II:ssa HP-lisäyksenä Intrinsic-kutsuja, joita hyväksikäyttäen TIPAssa onkin saatu näppärästi aikaan selkeitä ohjelmallisia ratkaisuja. Tiedostojen huono puoli MPE:ssä on niiden staattisuus, tiedoston ominaisuudet mm. koko, rivin pituus, jne. määritellään tiedoston luonnin yhteydessä, eikä näitä ominaisuuksia voida muuttaa ajoaikaisesti.

Tietokantaratkaisu on MPE:n Image, joka on tehokas suurten volyymien käsittelyssä. Image on verkkotietokanta, jossa haetaan tietoa joko osoitteella tai kokonaisella avaintiedolla, joita TIPAn eri tiedostoissa on yhdestä viiteen kappaletta. Imagessa on kolmen tyyppisiä tiedostoja: Detail-, automaattimaster- ja master-tiedostoja. Detail-tiedostoissa ovat tietokantaan talletetut tiedot ja master-tiedostoissa on erilaisia hakupolkuja avainhakuja varten. Yhdelle detail-tiedostolle voidaan määritellä useita master-tiedostoja.

Tietokannan tietoja voidaan lukita kolmella tasolla; Tietokantatasolla, tiedostotasolla ja tietuetasolla. Ohjelmallisesti lukitukset voidaan lukita joko ehdollisesti tai ehdottomasti. Ehdollisessa lukituksessa ohjelma jää odottamaan lukituksen vapautumista, jos tieto jo on käytössä. Ehdottomassa lukituksessa ohjelma jatkaa suoritustaan, vaikkei tietoa pystyttäisi lukitsemaan.

Imagen tietojen käsittelyyn on HP COBOLissa systeemialiohjelmat eli tietokantakutsut, jotka ovat TIPAssa suurimmaksi osaksi piilotettu ohjelmoijalta kopiokirjaston tietokannan käsittelyn tiedostojen avulla. Kopiokirjastossa on tietokannan käsittelyyn kirjasto kankas, jossa on kannan tietojen hakuun ja käsittelyyn liittyvät rutiinit. Tietokantarutiineihin kuuluvat tiedon haku, luvut eteen- ja taaksepäin, ketjuluvut eteen- ja taaksepäin, päivitys, lukitukset ja tuhoaminen.

Tulosteet ohjataan MPE:ssä erityiselle spooler-ohjelmalle, joka on monipuolinen työkalu tulosteiden hallintaan.

Näytönkäsittely on toteutettu VPLUS-työkalulla, jolla näytöt on "piirretty" ja niille on annettu ominaisuuksia, jotka mahdollistavat usean näytön näkymisen samaan aikaan kuvaruudulla. Vain yhtä näytöistä voidaan kuitenkin käsitellä kerrallaan. Näyttöjen kenttien ominaisuudet määritellään näytölle ja kenttiin syötetyille tiedoille voidaan tehdä VPLUS:ssa monimutkaisiakin tarkistuksia ja käsittelyjä, jolloin ohjelmien koodeissa tarkistukset voidaan jättää minimiin.

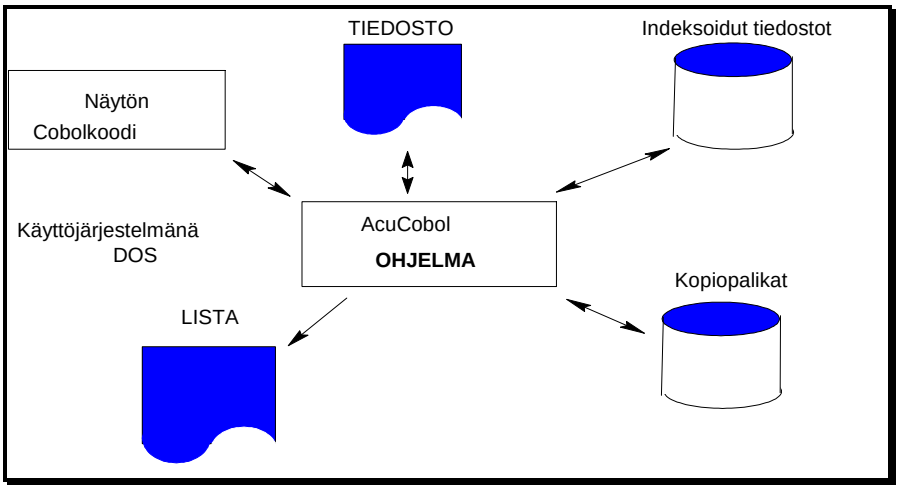
Erätyöt toteutetaan komentojonoilla käyttäen hyväksi MPE:n erätöiden käsittelyä.

Virheilmoitukset ovat TIPAssa virheviestitiedostossa joka on Catalog KSAM-tiedosto, eli HP3000:n indeksoitu tiedosto.

5.2.2. Kehityslaitteisto ja työkalut

Kehityslaitteena on PC, jossa on käytössä DOS ja Windows-käyttöjärjestelmät. Kuvassa 12 näkyvät kehityslaitteistossa käytettävät välineet. Kuvaa voi verrata kuvan 10

vastaaviin työkaluihin HP3000-ympäristössä. Indeksoitujen tiedostojen sijasta voitaisiin käyttää esimerkiksi MS SQL-tietokantaa tiedon tallennuspaikkana.



Kuva 12. Ohjelman ympäristö PC:llä

Kääntäjä on AcuCOBOL/85, COBOL-kääntäjä, joka tuottaa objektikoodia AcuCOBOL runtimelle. Objektikoodit ovat täysin siirrettäviä ja toimivat kuudessa käyttöjärjestelmässä ja yli 600 eri tietokoneella omalla runtime-versiollaan ajettuina. AcuCOBOL/85 on Ansi 85 kääntäjä, jossa on lisäksi useita ominaisuuksia tunnetuista COBOL-kääntäjistä sekä useita parannuksia mm. näyttöjen käsittelyyn aikaisempiin COBOL-versioihin verrattuna. AcuCOBOL/85 on koodattu C-kielellä ja runtimeen voidaan linkata omia C-kielisiä ohjelmia, jotka tämän jälkeen ovat käytössä aina kun AcuCOBOL/85-ohjelma käynnistetään.

Ohjelmakoodien esikäsittely HP COBOLin ja AcuCOBOLin välillä tehdään Windows-käyttöympäristössä Word-tekstinkäsittelyohjelman makroilla. Makrot ajetaan HP COBOL-ohjelmakoodeille ja niillä muokataan COBOL-koodia siten, että se on AcuCOBOLin syntaksin mukaista. HP COBOL-koodeista myös poistetaan MPE-käyttöjärjestelmän kutsuja ja muita käyttöjärjestelmän ominaisuuksia hyväksikäyttäviä koodin osia**.**

Ohjelmakoodien jatkokäsittely tehdään PC:n Qedit-editorilla, jolla tarkastetaan Word-makrojen korjausten oikeellisuus ja muutetaan koodia edelleen käyttöjärjestelmäriippumattomaksi**.**

Kopiokirjasto on yhdessä hakemistossa, jokainen kopioitava kokonaisuus on omassa tiedostossaan kutsunimellään.

Tiedostojen käsittelyä muutetaan siten, että kaikki tiedostot (ei tietokanta) ovat peräkkäistiedostoja. Ohjelmakoodeihin tehdyillä muutoksilla pyritään saamaan aikaan peräkkäistiedostoilla sama lopputulos kuin alkuperäisten ohjelmakoodien hyväksikäyttämillä erilaisilla tiedostomuodoilla.

Tietokantana käytetään AcuCOBOLin mukana tulevaa indeksoitujen tiedostojen järjestelmää nimeltään Vision. Vision on perustoiminnot sisältävä tietokantajärjestelmä, jota käytetään suoraan COBOL-kutsuilla.

Tulosteet ohjataan suoraan kirjoittimelle ajojen kirjoittamien komentojonojen avulla.

Näytönkäsittely tehdään COBOLilla generoimalla VPLUS-formilistauksista Word-makroilla COBOL-koodia, jota jatkokäsitellään vielä Qedit-editorissa.

Erätöitä ei DOS-käyttöjärjestelmä tunne, joten ajot ajetaan komentojonoilla heti käynnistyksen jälkeen.

Virheilmoitukset ovat PC:ssä virheviestitiedostossa joka on indeksoitu Vision-tiedosto.

5.2.3. Kohdelaitteisto ja työkalut

Laitteisto, johon objektikoodit siirretään, voi käyttää jotain AcuCOBOLin tukemia käyttöjärjestelmiä. Koneeseen hankitaan AcuCOBOL Runtime ja objektikoodit siirretään kohdekoneelle.

Erätyöt tehdään jokaisen käyttöjärjestelmän erätyökäsittelyn mukaisesti muokkaamalla ajojen käynnistyksien tekemiä komentojonoja kunkin käyttöjärjestelmän komentojonojen muotoisiksi.

Lajittelu erätöissä tehdään muodostamalla käyttöjärjestelmän omaa lajittelua hyväksikäyttävä ohjelma, joka tekee ajon käynnistyksen komentojonon tiedoista käyttöjärjestelmän komennon.

Tulosteet ohjataan kirjoittimille tai spooler-järjestelmälle komentojonoilla.

Tiedostot ovat peräkkäistiedostoja.

Tietokantana on AcuCOBOLin mukana tuleva Vision tai esim. VAX-koneissa niiden oma indeksoitujen tiedostojen järjestelmä, joka toimii kuten Vision. Tietojen tallennukseen voidaan käyttää myös muita kantaratkaisuja, jotka tukevat kannan käsittelyn tietokantarutiineja, joihin kuuluvat tiedon haku, luvut eteen- ja taaksepäin, ketjuluvut eteen- ja taaksepäin, päivitys, lukitukset ja tuhoaminen. Tietokantaratkaisuista käytössä ovat SQL-kannat, kuten MS SQL.

5.3. Siirrot

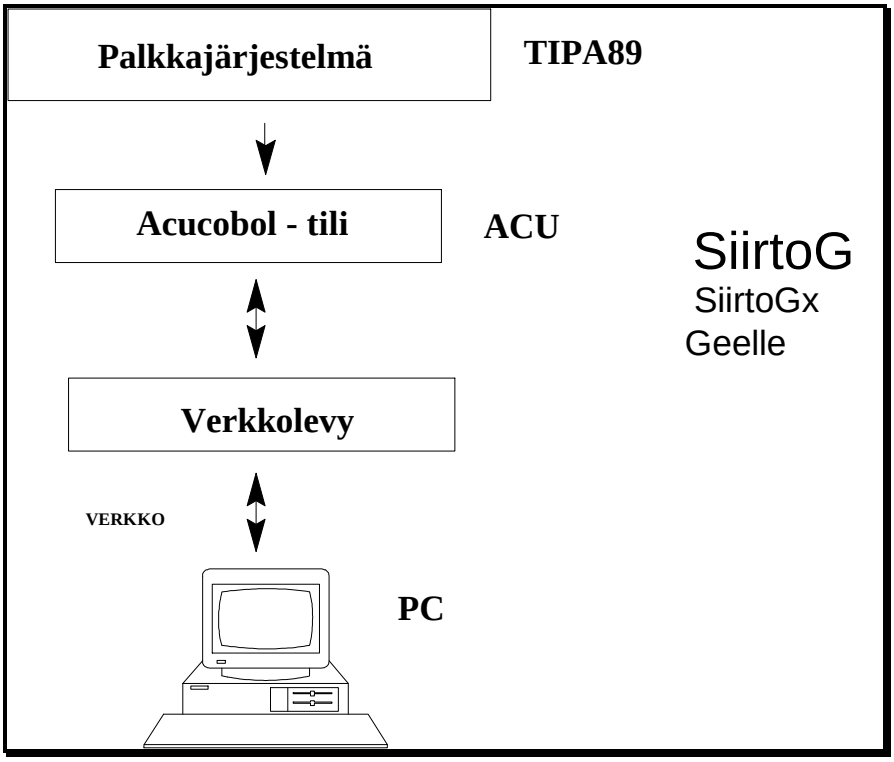
5.3.1. Tiedostosiirrot

Tiedostojen siirtoon on useita mahdollisuuksia. HP3000:n Resource Sharing on helposti automatisoitava tapa suurten massojen siirtoon MPE -tiedostoista lähiverkon levylle PC:n käytettäväksi. Tiedostojen jako PC-ympäristön ja HP3000-ympäristön kesken aiheutti muutamia merkkilajisto-ongelmia, jotka kuitenkin pystyttiin ratkaisemaan valitsemalla tiedostojen käsittelyssä käytettävät välineet, jotka eivät muokkaa tiedostojen sisältöä eli käsittelevät tietoa binäärisesti.

5.3.2. Ohjelmien siirrot

HP3000:lla olevat ohjelmat ovat TIPA89-tilissä SOURCE-ryhmässä. Sieltä ne siirretään MPEX-komentojonoilla ja Resource Sharing-ohjelmiston Discmgr-ohjelman avulla HP3000-koneen DOS-verkkolevylle.

Discmgr on HP3000:n työkalu, jolla voidaan käsitellä sen omalta levyltä jaettua DOS-tyyppistä verkkolevyä ja siirtää tiedostoja MPE- ja DOS-käyttöjärjestelmien välillä. PC:llä olevan Lan Manager-verkko-ohjelmiston avulla ohjelmat ovat PC:llä käytettävissä. Kuvassa 13 näkyy ohjelmien ja tiedostojen siirron kulku.



Kuva 13. Tiedostojen siirrot HP3000 ja PC:n välillä

HP3000-laitteessa on komentojono (SIIRTOG), joka siirtää tiedot ulos palkkajärjestelmästä. Komentojono tekee tiedostojen siirron TIPA89-tilistä ACU-tiliin ja sieltä verkkolevylle. Tiedostot siirretään hakemistoon \SIIRTO. Siirrettävät ohjelmakoodit ovat HP3000-laitteen tiedostossa. Komentojono SIIRTOG:

```
%RUN GOD.PUB.VESOFT
%YESPURGE @.SIIRTO.ACU
%COPY ^GEELLE.SOURCE.ACU,=.SIIRTO.ACU
%RUN DISCMGR.PPC.SYS,STDIN=SIIRTOGX
%RUN MORTAL.PUB.VESOFT
```

Siirtogx on komentojono Discmgr-ohjelmalle, jolla tiedostot viedään verkkolevylle. Komentojono SIIRTOGX:

```
MTD @.SIIRTO.ACU TO G:\ACUVARM\SIIRTO\*
RENAME G:\ACUVARM\SIIRTO\SP* SP*.TXT
EXIT
```

5.3.3. Näyttöjen siirrot

HP3000:lla olevat TIPAn näytöt ovat VPLUS-formitiedostoissa, josta niitä pystytään käsittelemään ja listaamaan. Listaus tapahtuu tiedostoon formlist, joka on ohjattu käyttöjärjestelmässä oletuksena kirjoittimelle. Näyttölistaus tulostetaan tiedostoon, joka kopioidaan ASCII-muotoiseen siirtotiedostoon. Siirtotiedosto tuodaan HP3000:n verkkolevylle kuten ohjelmakoodit tai siirretään Advancelink -pääte-emulaattorilla suoraan PC:n kovalevylle.

5.3.4. Kantojen siirrot

TIPAn tiedot ovat Image-tietokannassa, joka on saatavilla vain HP3000 -koneelle, joten tietojen siirto ei onnistu pelkkänä tiedostosiirtona.

TIPAan on tehty jo ohjelmistoa luotaessa ohjelmat tietojen siirtoon tietokannan ja ASCII-tiedostojen välillä. Näillä ohjelmilla kannasta siirretään ohjelmiston käytön vaatimat perustiedot mm. yleiset ohjaustiedot, joiden pohjalta ohjelmisto muokataan asiakkaan ympäristön mukaiseksi. Tietokannan tiedot siirretään ASCII-tiedostoina, samoin kuin ohjelmakoodit.

5.3.5. Muut siirrot

Virheviestitiedosto on tiedostossa ASCII-muodossa ja kopiokirjastotiedosto sekä sovelluskuvaukset, jotka myös ovat MPE-tiedostoja, siirretään kuten ohjelmakoodit.

5.4. Konversiot

5.4.1. Tiedostokonversiot

Tiedostojen muuttaminen AcuCOBOLille ja eri käyttöjärjestelmille on toteutettu erilaisilla tavoilla. Ohjelmakoodien ja näyttölistausten muokkaus AcuCOBOLille tapahtuu Basicilla toteutetuilla Word-makroilla ja editorilla. Virheviestitiedosto muutetaan Vision-tiedostoksi AcuCOBOL-ohjelmalla. Kopiokirjasto pilkotaan Word-makrolla erillisiksi include -tiedostoiksi.

5.4.2. Ohjelmien konversio

Windows WordBasic-makroja on tehty Wordin Template-tiedostoon yli 40 kappaletta. Makroista suuri osa muuttaa ohjelmakoodeja jollain tavalla. Esim. COPY REPLACING-lauseketta muutetaan HP COBOLin ja AcuCOBOLin eroavaisuuksien

takia. Muutama makro ohjaa muiden makrojen toimintaa ja luo käyttöliittymän ohjelmakoodeja muutamille makroille.

Kaikkia ohjelmakoodeja ei pystytä täydellisesti konvertoimaan makroilla, vaan konvertoidut tiedostot tarkastetaan editorissa ja tarvittavat muutokset tehdään, jotta koodi menee kääntäjästä läpi. Ohjelmien konvertoinnin ongelmat johtuvat suurelta osin ohjelmointistandardien puuttumisesta tai siitä, että standardeja ei ole noudatettu.

Ohjelmien konversion työkaluksi valittiin WordBasic, koska suuri osa tuotekehityksen ja ohjelmoinnin osaajista oli saanut peruskoulutuksen Windowsiin, Windows oli kehitysalustana muissa ohjelmistotuotteissa ja Wordia käytetään dokumentointiin. Vaihtoehtona oli Unixin makroilla, *sed*- ja *awk*-komennoilla, toteutettu konversiokirjasto. Näitä makroja tehtiinkin muutama kappale alkuvaiheessa, mutta niiden muuttaminen oli hankalaa ja osaamista oli vain muutamalla henkilöllä.

5.4.3. Kopiokirjaston konversio

Kopiokirjasto on yhdessä tiedostossa verkkolevyllä. Kopiokirjaston tiedostossa on yleisiä käsittelyrutiineja, joilla COBOLin saadaan aikaan samanlaisia ominaisuuksia kuin uudemmissa ohjelmointikielissä on käytössä esiprosessorimuuttujilla ja include-tiedostoilla. Käsittelyrutiinit sisältävät toiminnallisuuksia, joita käytetään useassa paikassa ja kopiokirjastosta näitä rutiineja voidaan käyttää parametroidusti halutun toiminnon suorittamiseksi.

Kopiokirjaston tiedosto jaetaan yksittäisiin kopiotietoihin Word-makrolla ja näihin tiedostoihin tehdään koodien muokkaukset samoin kuin muutokset muihinkin ohjelmakoodeihin. Kopiokirjastoon joudutaan kirjoittamaan uusia kopiotietoja lähinnä muuttamaan komentojen COPY REPLACING -lausekkeet tietokannasta löytyvillä oikeilla nimillä. Esimerkiksi KADHENKI on kannan käsittelyn KANKAS-kopiopalikan kopio, jossa ==$$== on korvattu DHENKILO-tekstillä.

5.4.4. Näyttöjen konversio

Näyttöjen formilistaukset ovat verkkolevyllä joko yksittäin tai yhdessä formitiedostossa, joka pitää pilkkoa näyttöjen listauksiksi Wordin makrolla. Makro tekee useita näyttöjä sisältävästä listauksesta näyttöjen nimillä erillisiä tiedostoja, joissa on näyttöjen konversioon käytettävän makron vaatimat tiedot.

Näyttöjen listauksien konversio tapahtuu kahdessa osassa. Ensimmäisessä osassa luetaan listaukselta näytön kenttien määrittelyt ja ominaisuudet niistä tehdään kenttien COBOL-kuvaukset. jotka toisessa vaiheessa muutetaan makroilla lopulliseksi käännösvalmiiksi COBOL-koodiksi. Näyttöjen ohjelmalistaukset ovat liitteessä 1.

Kaikki VPLUS-kutsut ohjelmissa ovat konversion jälkeen ennallaan. Näyttöjä kutsuvat ohjelmat kutsuvatkin nyt vain COBOL-ohjelmia, jotka ohjelman välittämien parametrien mukaan kutsuvat näyttöjen koodeja. Näytön koodit matkivat VPLUS-ohjelman toimintoja ja tekevät syötetylle datalle samat tarkastukset ja muunnokset kuin

VPLUS ja palauttavat datan ohjelmille. Ohjelmakoodeihin ei täten ole tarvinnut tehdä muutoksia näytönkäsittelyn muuttumisen takia.
Konvertoidut tiedostot tarkastetaan editorissa ja tarvittavat muutokset tehdään, jotta näytöillä tehdään tarvittavat tarkistukset ja COBOL-ohjelma toimii kuten VPLUS-ohjelmat HP3000:lla.

5.4.5. Virheviestitiedoston konversio

Virheviestitiedostoa joudutaan muuttamaan editorilla koska virheilmoituksissa on käytetty hyväksi MPE:n ominaisuuksia ja nimiä. TIPAssa on noin 300 alle 80 merkin mittaista virheilmoitusta, jotka tulostetaan tekstinä näytön alalaitaan syöttökenttien alapuolelle. HP3000 -koneella tekstit ovat ASCII-muodossa editoitavia tekstejä, jotka *gencat*-ohjelmalla muutetaan MPE:n indeksoiduiksi tiedostoiksi eli catalog-tiedostoiksi.

Virheviestit muutetaan Unixissa ja DOSissa Vision-tiedostoon COBOL-ohjelmalla *catalog* antamalla komento *runcbl catalog*. Virheviestien näyttäminen tapahtuu ohjelmissa samalla tavalla kuin aiemminkin, vain virheviestien käsittelyä on muutettu siten, että viestit haetaan joko Vision-tiedostoista tai käytettävästä tietokannasta.

5.4.6. Tietokantojen ja tietokantakäsittelyn konversio

Tietokannan tiedostot ovat HP3000:lla Image-tiedostoja, joiden tieto on sekä ASCII, että binäärimuotoista. Tiedostojen kuvaukset saadaan Imagesta listauksina, joista näkyvät tiedoston kentät, niiden pituudet ja tyypit sekä avainkentät. Ohjelmissa on käytetty tietokannan käsittelyssä Imagen intrinsiikkikutsuja, jotka ovat, kuten näyttöjen VPLUS-kutsutkin, koodattu COBOLilla Unix- ja DOS-ympäristöissä. Imagen intrinsiikkikutsut koodattiin COBOLilla ja näihin COBOL-ohjelmiin rakennettiin Imagen sisäisten muuttujien mukaisesti yhteinen data-alue. Muuttujien avulla saatiin TIPAn ohjelmiin koskematta rakennettua Imagea vastaavat virhetilanteiden käsittelyt ja tietokantojen monipuolinen toiminta.

Tiedostojen kuvauksien siirto COBOL-kuvauksiksi on suoraviivaista ja avainkenttien määrittely indeksoiduille tiedostoille tehtiin kopiokirjaston tiedostoihin, joita kannankäsittelyn ohjelmat käyttävät. Visionin vaatimus yhdestä tietueen yksilöivästä avaimesta indeksoidussa tiedostossa ei kaikissa Image-tiedostoissa toteutunut, vaan indeksoituja tiedostoja varten luotiin tekninen avain, joka yksilöi tietueet.

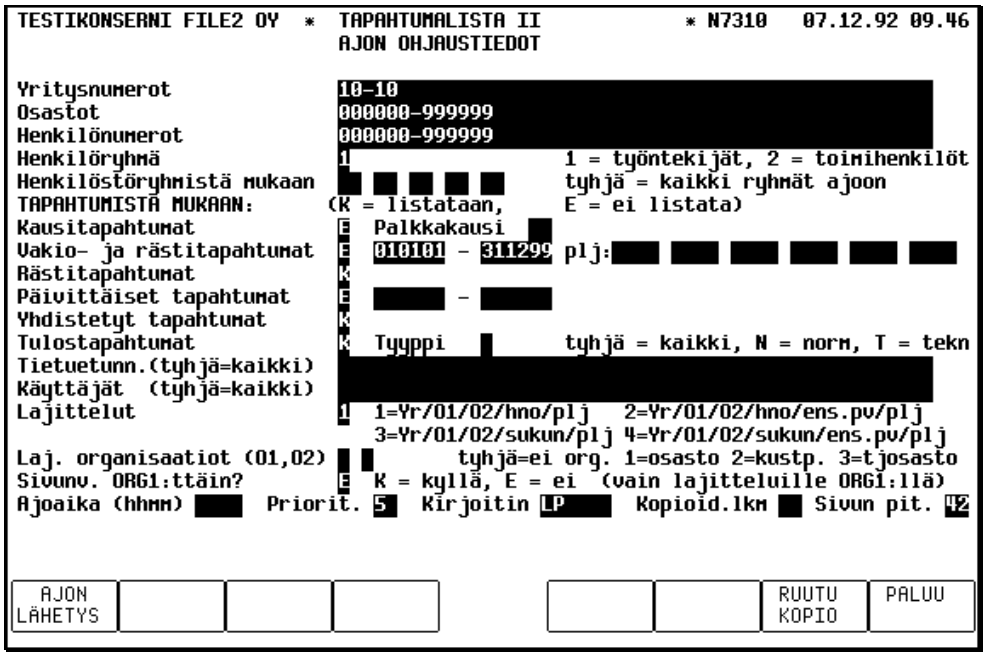
5.5. Erätyöt

Erätöiden käsittelyssä on ratkaistu jokaisen käyttöjärjestelmän omilla erätöiden käsittelyyn tehdyillä ratkaisuilla. Pyrkimys on jälleen se, ettei olemassa olevia ohjelmia muuteta, vaan muutokset tehdään kopiokirjastossa oleviin tiedostoihin.

Erätöiden käynnistysnäytöillä annetaan ajojen parametrit, jotka HP3000-koneilla välitetään erätöille jobikorteilla. Jobikortit ovat MPE:n komentojonoja.

Käynnistysohjelmat suorittavat aina kopiotiedoston ajojono. Ajojono-kopiotiedosto käynnistää erätyön. Tähän kopiotiedostoon tehtävillä muutoksilla saadaan eri käyttöjärjestelmissä työt ajastettua, jos se on mahdollista käyttöjärjestelmässä.

Muutokset erätöiden käsittelyyn on saatu rajoitettua yhteen kopiokirjaston osaan ja sen kutsumaan käyttöjärjestelmäkohtaiseen komentojonon tekevään ohjelmaan. Erätöiden käynnistys on täten tullut raskaammaksi, mutta tällä ei liene merkitystä, koska ajastettavia ajoja ei erätöiden luonteen mukaan ole tarkoitus saada käyntiin mahdollisimman nopeasti. Kuvassa 14 on tapahtumalistan erätyön käynnistysnäyttö, jota käytetään joko eräajon käynnistämiseen tai eräajopyynnön lähettämiseen.



Kuva 14. Erätyön käynnistysnäyttö, N7310

TIPAn erätyön toiminnallisuuden kuvauksessa käytetään esimerkkinä ohjelmaa tapahtumalista II, joka on TIPAlle tyypillisesti nimetty siten, että toiminnallisuudella on numero, jota käytetään kaikissa tähän ohjelmaan liittyvissä toiminnoissa ja ohjelmaan liittyvien tiedostojen nimeämisessä. Liitteessä 2 on yksityiskohtaisempi selvitys eräajon toiminnallisuudesta ja syntyvistä tiedostoista.

TIPAn erätyön käynnistysohjelma (*N7310*) kirjoittaa jobikortin (JP7310 -tiedoston), jota käytetään erätyön käynnistyksen parametritietona. Erätyön käynnistävässä ohjelmassa (*PP7310A*) kutsutaan ajojonon käynnistystä, joka sijaitsee ajojono -kopiotiedostossa. Tässä kopiotiedostossa kutsutaan käyttöjärjestelmästä riippuen erilaista erätyöohjelmaa. DOS- ja Windows-käyttöjärjestelmissä ohjelmaa *PPDOSERA* ja Unix-käyttöjärjestelmässä ohjelmaa *PPUNXERA*. Erätyöohjelma lukee jobikorttia ja tekee siitä käyttöjärjestelmän ymmärtämässä muodossa olevan komentojonon; DOS-laitteilla .bat-tiedoston ja Unixilla scriptin. Lisäksi erätyöohjelma tekee input-tiedoston INP7310 ohjelmille *POIMI2* ja *PP7310*, jota nämä ohjelmat

lukevat. Parametrit eivät siis Unixissa ja DOSissa ole komentojonossa, vaan erillisessä input-tiedostossa, joka ohjataan erätöille komentojonon ajokomennossa.

Kuvassa 15 on komentojono, jolla ajetaan tapahtumalista-ohjelma ja tulostetaan tapahtumalista halutulle kirjoittimelle. Unix, Windows ja VMS huomioivat ajastuksen, kun taas DOSissa työ ajetaan heti.

```
runcbl -i INP7310 -o STD7310 -l -e STD7310 PPOIMI2
runcbl -i INP7310 -o STD7310 -l -e STD7310 PP7310
DOS
print \D:LP   LP7310
print \D:LP   PL7310
del dhenkilo
del inp7310
Unix
lp -dlaser -n01 LP73101215
lp -dlaser -n01 P   L73101215
rm dhenkilo
rm inp73101215
```

Kuva 15. Komentojono eräajon ajamiseksi

Palkka- ja henkilöstöhallinnon tietoja käsiteltäessä tulee ottaa huomioon henkilöiden valtuuksien vaikutus listoilla näkyvään tietoon. Tätä silmälläpitäen välitetään parametritiedostoissa myös kryptatut valtuustiedot, jotka tarkastetaan jokaisessa eräohjelmassa samalla tavalla kuin ohjelmistoa käynnistettäessä.

5.6. Lajittelu

TIPAssa erätyöt käyttävät tiedostojen lajitteluun HP3000:lla käyttöjärjestelmän sort-ohjelmaa, joka saa komentonsa jobikortilta. Esimerkkikuvassa 16 on lajittelu dhenkilo-tiedostolle.

```
!RUN SORT.PUB.SYS
INPUT DHENKILO
OUTPUT LAJIT
KEY 011,002,BYTE
KEY 003,006,BYTE
END
```

Kuva 16. Lajittelun ajava ohjelma

Sort käynnistetään, ja se lukee seuraavat rivit aina end-riviin saakka. Ensin nimetään lajiteltava tiedosto ja tulostiedosto. Lajitteluavaimet ovat key-riveillä. Ensimmäinen numero tarkoittaa avaimen alkupositiota ja toinen sen pituutta.

Lajittelu muutettiin konversiossa ensimmäisessä vaiheessa COBOL-ohjelmaksi, jota kutsutaan erätöiden komentojonoissa kuten erätöitä ja henkilöiden poimintaohjelmaakin. Lajittelu on COBOL-ohjelmana selvästi käyttöjärjestelmän lajittelua hitaampi, mutta ohjelmiston tehokkuuteen tällä ei ole huomattavaa vaikutusta,

koska lajittelua käytetään ainoastaan erätöissä, ja selvästi suurimman osan näiden ajojen viemästä ajasta käyttää raskas laskenta tai tulostustoiminta.

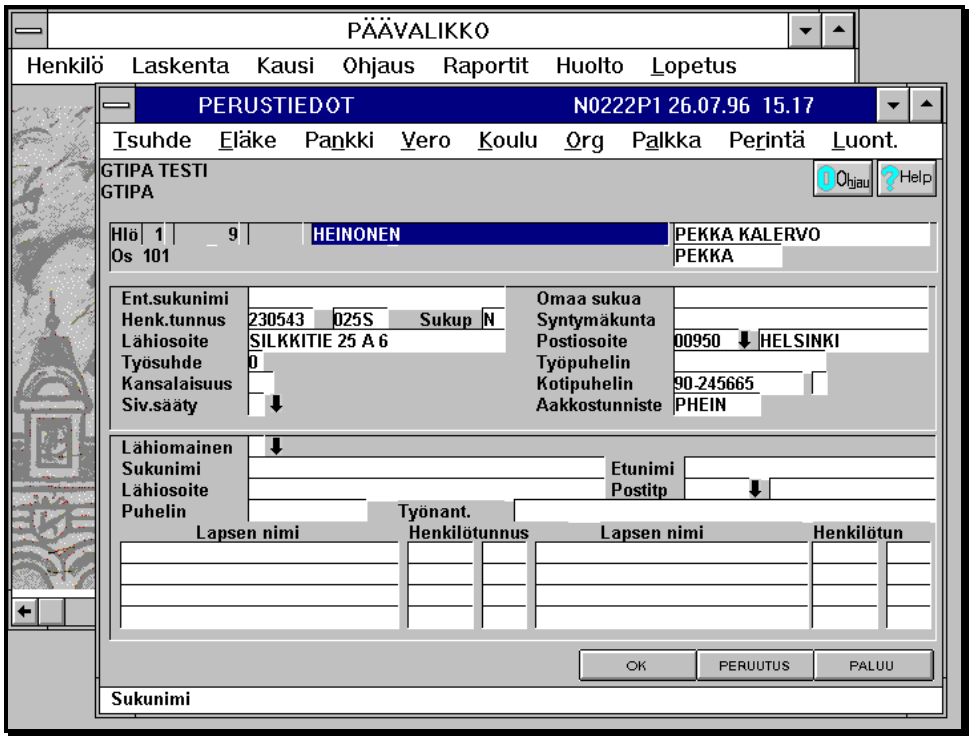
Vaihtoehdoksi COBOL-kieliselle hitaammalle lajittelulle toteutettiin C-kielinen lajitteluohjelma, jolla saatiin suurien massojen käsittelyä nopeutettua ja voitiin ajaa TIPAa hitaassakin laitteistossa.

5.7. Siirrettävä TIPA

TIPA on tällä hetkellä järjestelmä, jonka toimii niin suljetussa HP3000-ympäristössä, kuin avoimessa ympäristössä useissa käyttöjärjestelmissä. Eri Unix-, Dos-, Windows-käyttöjärjestelmäversiot sekä VAX/VMS-ympäristö ovat mahdollisia käyttöjärjestelmiä. Laitteistoja, joilla TIPA toimii, useita satoja sisältäen kaikki suosituimmat laitemerkit ja viikon toimitusajalla mihin tahansa laitteeseen, joissa käytetään tuettuja käyttöjärjestelmiä.

Tietokantavaihtoehtoina ovat avoimessa ympäristössä indeksoidut tiedostot ja tietokantaratkaisut kuten Informix ja Oracle. Lisäksi AcuCOBOL tukee MS SQL-yhteensopivia tietokantoja.

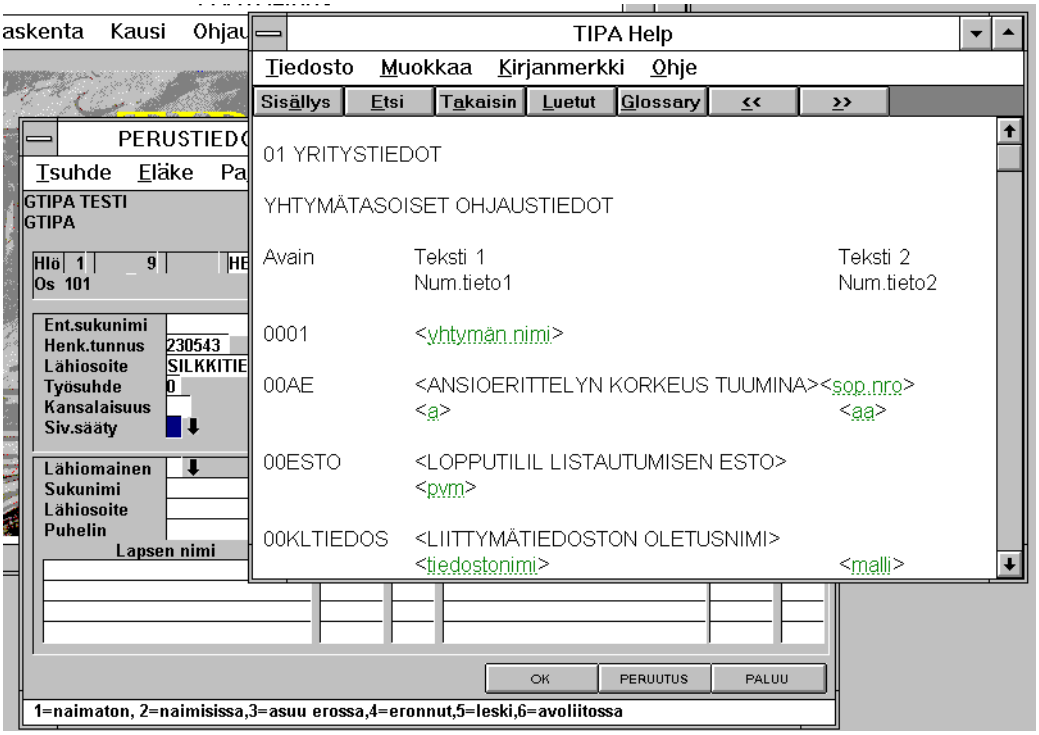
Näytönkäsittelyyn on Windows-ympäristössä vaihtoehto merkkipohjaiselle käsittelylle. Perusohjelmiston pienillä muutoksilla, lähinnä näytönkäsittelyn rajapintaan, ohjelmistossa voidaan käyttää kuvassa 17 näkyvää graafista käyttöliittymää, joka tuo tullessaan paljon uutta. Valikkotoiminnallisuutta on muutettu ja graafisen käyttöliittymän myötä on voitu toteuttaa näyttöihin kenttätarkastuksia ja lisätä käyttöliittymän vuorovaikutusta.



Kuva 17. Graafisen TIPAn päävalikko ja perustietonäyttö

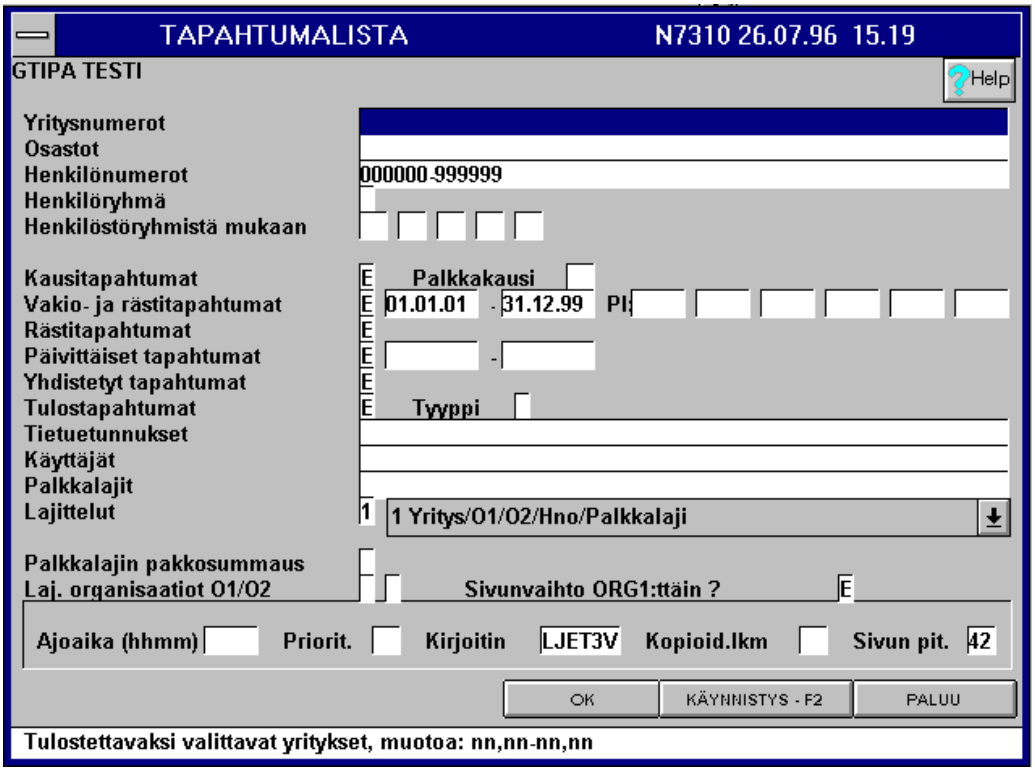
Graafisessa käyttöliittymässä TIPA toimii samaan tapaan näppäimistöltä käytettynä kuin merkkipohjainen järjestelmäkin. Käytettävät ohjelmat ovat samoja, ainoastaan näytönkäsittelyt ovat omia ohjelmiansa. Uusien näyttöjen koodi keskustelee perusohjelmiston kanssa saman merkkipohjaista tietoa välittävän rajapinnan kanssa kuin avoimen ympäristön AcuCOBOL-koodi ja HP3000-ympäristön VPLUS-sovellus.

Uutta TIPAssa ovat graafisuuden lisäksi selailut valinnaisissa kentissä ja Windowsin online-aputoiminto. Aputoiminto käyttää TIPAn sovelluskuvausta, jota nykyään ylläpidetään MS Wordissä. Word-dokumentti ajetaan Windowsin help-generaattorin läpi ja liitetään ohjelmistoon, jolloin ne näkyvät kuvan 18 mukaisesti. Lisäksi sovelluksessa on laajojen ohjaustietojen aputoiminto, joka helpottaa ohjelmiston parametrointia ja käyttöä. Kenttäkohtaiset aputekstit näkyvät graafisessa versiossa näytön alalaidassa.



Kuva 18. Graafisen TIPAn aputoiminto

Ohjelmiston konversion esimerkkinä olleen ohjelman, tapahtumalistan graafisessa versiossa (kuva 19) ei ole suuria eroja merkkipohjaiseen versioon verrattuna. Lajittelun voi valita selailusta ja näytössä on aputoiminto. Koska ohjelma on pelkkä käynnistysnäyttö, eivät muutokset toiminnallisuudessa ole perusteltuja. Yllä olevissa kuvissa olevaan perustietojen näyttöön on sitä vasten tehty myös toiminnallisia muutoksia ja graafisessa versiossa mahdolliseksi tulleita lisäominaisuuksia on käytetty hyväksi. Henkilön perustietonäytöltä voidaan siirtyä painonappien avulla muille henkilön tietoja sisältäville näytöille.



Kuva 19. Graafisen TIPAn tapahtumalistan käynnistys

6. YHTEENVETO

Ohjelmistojen siirrettävyyden edut ovat selkeitä ja perusteltuja ja ratkaisuja siirrettävyyden toteuttamiseen on useita. Korkean tason ohjelmointikielistä paljon käytetyt ovat vahvan tukensa ja yleispätevyytensä johdosta suositeltavia vaihtoehtoja. Ohjelmointikielten eri murteet saattavat kuitenkin kehittyessään erota standardeista ja näiden epästandardien ominaisuuksien välttämiseksi tulee seurata tarkasti kielen yleisiä kehityslinjoja.
Siirrettävyyden kanssa on kamppailtu aina siitä lähtien, kun ensimmäinen muutos tehtiin laitteeseen tai käyttöjärjestelmään. Kymmeniä vuosia siirrettävyyden tutkimista ja ratkaisumallien etsimistä ei ole tuonut toivottua tulosta. Laitteistojen ja järjestelmien nopea kehitys aiheuttaa sen, että siirrettävyys tulee aina kehityksen askelten jäljessä. Totuushan on, ettei kehitystä tapahtuisi, jos kaikki kehitys olisi standardoitu. Kehityksen standardoiminen kuitenkin osaltaan helpottaa siirrettävyyden ongelmissa ja standardeja luotaessa otetaan jo huomioon siirrettävyyden myöhemmin aiheuttamat kustannukset.

Ainoana mahdollisuutena kattavan standardin luontiin tuntuu olevan se, että standardoitavan työkalun/työympäristön toimittajan täytyy olla suuri yritys tai yhteenliittymä ja standardin mukaisten työkalujen, sovellusten ja tuen tulee olla halpaa tai ilmaista ja niitä kaikkia tulee olla helposti saatavilla. PC-markkinoiden ohjelmistojen jättiläinen Microsoft on valtavalla markkinoinnillaan saanut kasvatettua markkinaosuutensa sellaiseksi, että sillä on mahdollisuus määritellä kattavia standardeja. Vielä eivät laitteistojen ja järjestelmien tehot ja ominaisuudet kuitenkaan riitä suurten ohjelmistojen ja tietokantojen käsittelyyn, joten vielä tarvitaan erilaisia käyttöjärjestelmiä ja laitteita toimimaan yhteistoiminnassa tietokoneverkkojen välityksellä.

Olemassa olevan ohjelmiston kirjoittaminen uudelleen olemassa olevalla siirrettävällä ohjelmointikielellä kuten Javalla on huomioitava vaihtoehto. Uudelleenkirjoituksen työ on huomattavasti pienempi kuin ohjelmiston toteuttaminen alusta lähtien ja nykyaikaisilla toteutustavoilla voidaan modulaarisesti toteutettu kohtuullisen laadukas ohjelmisto korvata osissa.

Ohjelmistokehittimet, kuten Progress, ovat ratkaisuna nykyaikaisia, mutta usein niiden ongelmana on sitoutuminen yhteen kehitysympäristöön ja sen tarjoamiin työkaluihin. Esimerkiksi Progress on versiosta 7 lähtien versioita, joilla voidaan tehdä Windows-sovelluksia. Kuitenkin siirtyminen Progressin versiosta 6 on työlästä ja kehittimen vaatimat laitteistoresurssit ja korkea hinta rajoittavat mielenkiintoa. Lisäksi

kehittimien yhteensopivuudesta vanhempien kehitinversioiden kanssa ei koskaan ole takuita.

Microfocus COBOL siirrettävine ympäristöineen on toimiva ratkaisu, koska jopa käyttöjärjestelmä on silloin Microfocus COBOL. Tällainen sitoutuminen yhteen järjestelmään ei liene kuitenkaan järkevää nopeasti kehittyvän tietotekniikan alalla. Perinteiset ohjelmointikielet eivät aina tue nykyaikaisia käyttöliittymiä eivätkä tietokantoja ilman merkittäviä muutoksia tai lisätuotteita.

AcuCOBOL on kehittyvä ohjelmointikieli, joka on syntaksiltaan perinteinen COBOL-kieli, mutta jolla on kehittyneitä ominaisuuksia ja useita lisätuotteita, joilla saadaan koodi erotettua rajapinnan taakse esimerkiksi näytönkäsittelyssä.

Siirrettävyyden avulla suunnittelijoiden ammattitaito ja useiden henkilötyövuosien työ on saatu Tietonauha-yhtiön palkka- ja henkilöstöhallinnon järjestelmän tapauksessa hyödynnettyä uuteen tuotteeseen. Kehityslaitteena on edelleen HP3000, jolla myös suurin osa asiakkaista tällä hetkellä toimii. Ohjelmistopäivitykset HP3000:lta toimivat sujuvasti ja kehitystyön painopiste on siirtynyt pikkuhiljaa pois HP3000:lta. Uutta koodia järjestelmän eri asennuksissa vaativat ainoastaan järjestelmään lisättävät uudet ominaisuudet. TIPA toimii nykyään useissa tietokoneissa AcuCOBOL runtimen avulla. Runtimen hinta vaihtelee eri laitteilla, mutta on selvästi kehittimien vastaavia hintoja halvempi.

Tietokantaratkaisut ovat lähes vapaasti valittavissa, joko muuttamalla kannankäsittelyn liittymäpintaa tai ottamalla käyttöön Acu4GL, joka tarjoaa tietokantakäsittelyt esim. Oracle-kantaan, jolloin COBOL-ohjelmasta voidaan käyttää tietoa myös tästä suositusta relaatiokannasta.

Näytönkäsittely on AcuCOBOLissa paljon vanhoja COBOL-murteita monipuolisempaa. AcuCOBOLin Windows-versio mahdollistaa COBOL-näytön käsittelyssä hiiren käytön ja Windows-tyyppiset valikot. Nämä työkalut on rakennettu siten, että itse COBOL-koodi on mahdollisimman paljon standardien mukaista, mutta toiminnallisuus toteutetaan ajoympäristössä omalla AcuCOBOLin tuotteella. Tärkeää on myös mahdollisuus dynaamiseen tiedonsiirtoon eri Windows-ohjelmien kanssa ja mahdollisuus rakentaa käyttöliittymä jollain Windows-käyttöliittymätyökalulla, kuten Visual Basicilla.

Kehittyminen Windows - Unix -akselilla helpottaa varmasti tulevaisuudessa ohjelmien siirrettävyyttä. Tulevaisuudessa tuki jommallekummalle tai molemmille käyttöjärjestelmille on useissa tietokoneissa ja lähitulevaisuudessa julkaistavat monikäyttöjärjestelmäiset tietokoneet mahdollistavat TIPAn käytön useissa erilaisissa tietokoneissa ilman muutoksia.

Tällä hetkellä valmiina olevia, suuria perinteisillä ohjelmointikielillä tehtyjä ohjelmistoja ei kannata jättää syrjään miettimättä voidaanko vanhaa ohjelmaa "ehostaa" mm. graafisella käyttöliittymällä tarvitsematta koodata koko sovellusta uudelleen.

LÄHTEET

Brown, P. J. ja Chambers, F: B. ja Griffiths, M. ja Lorho, D., "CNRC/SRC Software Portability Study", Software Portability: An Advanced Course, ed. P.J. Brown, Cambridge University Press, 1977

Curtis, William "Management and Experimentation in Software Engineering", IEEE, vol. 68, no. 9, September1980

Crosby, Philip B., "Quality is free", McGraw-Hill Book Company, 1979

Ghezzi, Carlo ja Jazayeri, Mehdi, "Programming Language Concepts", 1982

Halstead Maurice H., Elements of Software Science (Operating and programming systems series), Elsevier Science Inc., 1977

Henderson, John, "Software Portability", Gower Technical Press, 1988

Lecarme, Olivier (et al), "Software Portability with Microcomputer Issues", McGraw-Hill Publishing Company, 1989

Marttila, Petri, "Organisaatioiden välisen tiedonsiirron toteutus eräässä LVSI- alan tukkukaupassa", Diplomityö, Konetekniikan osasto, Tampereen teknillinen korkeakoulu. Tampere, 1990

Numerical Algorithms Group. 2010. The NAG Fortran Library. [WWW]. [Viitattu 17.3.2010]. Saatavissa: http://www.nag.co.uk/numeric/FL/FLdescription.asp

Pressman, Roger S., "Software Engineering; A Practitioners Approach, McGraw-Hill Publishing Company, 1990

Richards, M, "Portable Compilers", Software Portability: An Advanced Course, ed. P.J. Brown, Cambridge University Press, 1979

Schulmeyer, G. Gordon ja McManus, James I., Handbook of Software Quality Assurance, Van Nostrand Reinhold Company, 1990

Tausworthe, Robert C., "Standardized Development of Computer Software, Part III, Standards", Prentice Hall, 1981

Waite, William M., Janus: an implementation of UNCOL. In Proc. Eighth Hawaii Intl. Conf. on System Sciences, pages 207–209, 1975.

Wallis, Peter J. L., “Portable Programming”, the Macmillan Press ltd., 1982

LIITE 1, ESIMERKKI NÄYTÖN KONVERSIOSTA

Näytön konversiot toteutettiin generoimalla makroilla aiemmin käytössä olleesta ohjelmakoodista ja kuvaustiedostoista COBOL-koodia, joka toteutti AcuCOBOLissa samanlaisen näytön ja toiminnallisuuden kuin HP COBOLilla ja VPLUS-näytönkuvauksella saatiin toteutettua MPE-käyttöjärjestelmässä.

ALKUPERÄISEN NÄYTÖN KOODIA JA NÄYTÖN KUVAUSTIEDOSTOJEN SISÄLTÖÄ

Näytön otsikkotiedot

```
Form: N0050

   Repeat Option: N

   Next Form Option: C
   Next Form: $HEAD

   Reproduced from:

Comments:  Yleiskorotusten siirto kantaan
```

Näytön kuvaus, josta selviävät tekstien ja syöttökenttien paikat

```
********* ********* ********* ********* ********* ********* ********* **********
+YRITYS.................. *    YLEISKOROTUSTEN PÄIVITYS   * N0050   PVM..... KELLO
 ________________________                                  _________ _____
+ALAYKSIKKO.............
 ________________________

+Yritysnumerot               Y1 - Y2
                             __   __
+Henkilöryhmä                H        1 = työntekijät, 2 = toimihenkilöt
                             _
                                      tyhjä = työntekijät ja toimihenkilöt
+Yleiskorotustiedoston nimi  TIEDOSTO
                             ________
 Tuhotaanko työtiedosto
+levyltä päivityksen jälkeen? T       K = kyllä, E = ei tuhota
                             _
+Päivitettyjen henkilötietueiden lkm  KPL. kpl
                                     ____
+                                        Ovatko näytön tiedot oikein (K/E)? X
                                                                            _
-********* ********* ********* ********* ********* ********* ********* **********
```

Funktionäppäinten nimitekstit

```
Form Function Key Labels:
 ---f1---  ---f2---  ---f3---  ---f4---  ---f5---  ---f6---  ---f7---  ---f8---
                                                    RUUTU     PALUU
                                                    KOPIO
 --------  --------  --------  --------  --------  --------  --------  --------
```

Syöttökenttien kuvaukset

```
Field: YRITYS
   Num: 1    Len: 23    Name: YRITYS           Enh: NONE  FType: D  DType: CHAR
   Init Value:    0
Field: PVM
   Num: 2     Len: 8      Name: PVM                  Enh: NONE  FType: D  DType:
CHAR
   Init Value:
                                ...
                                ...
                                ...
Field: Y2
   Num: 52   Len: 2     Name: Y2               Enh: HI    FType: P  DType: DIG
   Init Value:
*** PROCESSING SPECIFICATIONS ***
JUSTIFY RIGHT
FILL LEADING "0"
IF EQ $EMPTY THEN
   FAIL               "Anna yritysnumeroväli                   " 0
Field: H
   Num: 53   Len: 1     Name: H                Enh: HI    FType: O  DType: CHAR
   Init Value:
*** PROCESSING SPECIFICATIONS ***
IN "1":"2"            "Valitse siirrettävä henkilöryhmä         "
                                ...
                                ...
                                ...
Field: X
   Num: 46   Len: 1     Name: X                Enh: HI    FType: P  DType: CHAR
   Init Value:
*** PROCESSING SPECIFICATIONS ***
IN "K","E"            "Ovatko näytön tiedot oikein? Vastaa K tai E."
```

ESIMERKKI KONVERTOIDUN NÄYTÖN COBOL-KOODISTA

```
Identification Division.
     ****************************************************************
     **  GRAMMER - Tool for converting VPLUS - formfiles to AcuCOBOL *
     **            Version 1.5 , 11.10.93  , TPo                     *
     **            Copyright 1993 Tietonauha-yhtiö OY, Timo Pokkinen *
     **            Konversio ajettu : 28.10.93 kello 12.45           *
     ****************************************************************
     *  Yleiskorotusten siirto kantaan

Program-id. N0050.

data division.
working-storage section.
```

VPLUS:n sisäisiä muuttujia COBOLilla

```
   copy SYS-WS.

01 this-form                  pic x(10) value "N0050".

01 repeat-option              pic 9(4) value 0.
01 nf-option                  pic 9(4) value 0.
01 cfnumlines                 pic 9(4)  value 23.
01 next-form                  pic x(10)  value " ".
```

Funktionäppäinten tekstit

```
01 label-buffer.
  02 filler    pic x(8) value space.
  02 filler    pic x(8) value " RUUTU  ".
  02 filler    pic x(8) value " KOPIO".
  02 filler    pic x(8) value " PALUU".
  02 filler    pic x(8) value space.

01 data-buffer.
  02 YRITYS    pic  x(23).
  02 PVM       pic  x(8).
  02 KELLO     pic  x(5).
  02 ALAYKSIKKO       pic  x(23).
  02 Y1        pic  9(2).
  02 Y2        pic  9(2).
  02 X  pic  x(1).
01 data-buffer-size   pic 9(4) value  78.
01 no-of-input-fields  pic 9(4) value  6.


linkage section.

   copy SYS-LNK.
   copy COMAREA.

screen section.
```

Näytöllä olevat tiedot on määritelty kahdessa osassa. Ensimmäisessä osassa (fixed-screen) ovat kaikki näytön staattiset tiedot, eli tekstit. Toisessa osassa (variable-screen) määritellään näytön muuttuvat tiedot. Näytön muuttuvissa tiedoissa ovat kuvattuina kaikki ne kentät, joiden tiedot päivitetään tietokannasta.

Näytön tekstit

```
01 fixed-screen.
  02  line 1     col 24 value " *    YLEISKOROTUSTEN PÄIVITYS".
  02             col 54 value "   * N0050   ".
  02  line 4     col 1 value "Yritysnumerot                ".
  02             col 32 value " - ".
  02  line 6     col 1 value "Henkilöryhmä                 ".
  02             col 31 value "        1 = työntekijät, 2 = t".
  02             col 61 value "oimihenkilöt".
                            ...
                            ...
                            ...
  02             col 31 value "              Ovatko näytön ti".
  02             col 61 value "edot oikein (K/E)? ".
```

Näytöllä olevat kentät

```
01 variable-screen.
  02  line 1     col 1     pic  x(23)
                            from YRITYS in data-buffer
                            auto.
  02             col 67     pic  x(8)
                            from PVM in data-buffer
                            auto.
                              ...
                              ...
                              ...
  02  line 23    col 80     pic  x(1)
                            using X in data-buffer
                            auto low reverse.
   copy PR-MAIN.
```

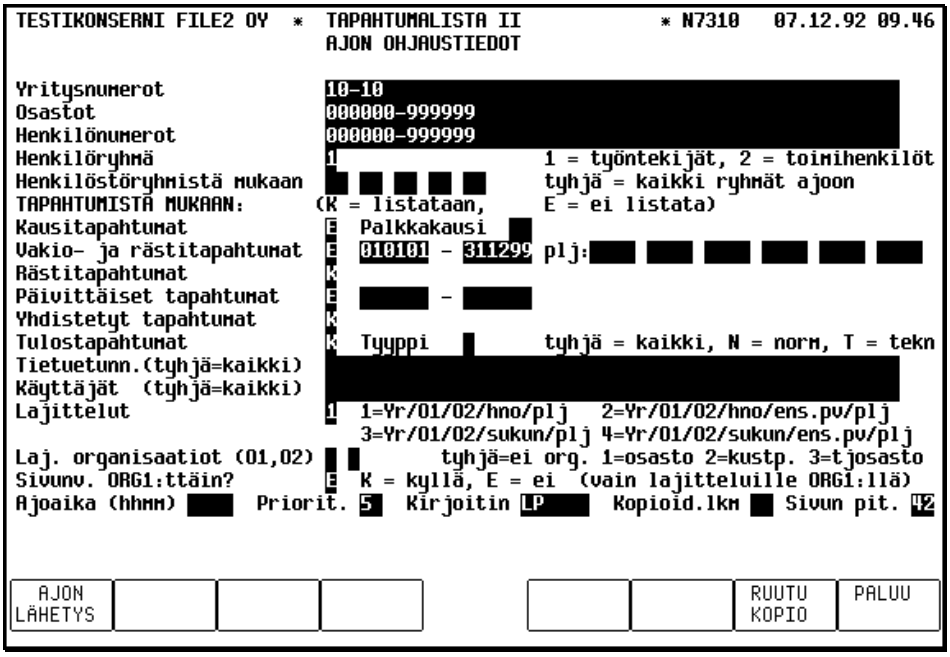
Kenttien tarkistukset

```
      copy PR-VFES1.
Y1-check.
   FILL LEADING "0"
   IF EQ $EMPTY THEN
                  to display-message
            go to Y1-check-fail.
  go to Y1-check-ok.
  Y1-check-fail.
    move 51 to field-index
    perform error-handling
    display Y1 in data-buffer
       line ln + 4 col 30 blink
    accept  Y1 in data-buffer with conversion
       line ln + 4 col 30
    on exception
      perform get-key
      if function-key
        go to section-exit
      else
        if help-print
          perform utilities
        end-if
      end-if
    not on exception
      accept crt-status-1 from escape key
    end-accept
    go to Y1-check.
  Y1-check-ok.
    if field-error (51) = 1
      perform clear-error
      display Y1 in data-buffer
         line ln + 4 col 30 low reverse.
  X-check.
     if X in data-buffer not = "K" and not = "E"
   IN "K","E"            "Ovatko näytön tiedot oikein? Vastaa K tai E."
                  to display-message
            go to X-check-fail.
  go to X-check-ok.
...
...
 X-check-fail.
    move 46 to field-index
     copy PR-VFES2.
     copy PR-VGB.
     copy PR-VGF1.
     evaluate fieldnum
           when 1     move YRITYS in data-buffer
                         to YRITYS in user-buffer
           when 2     move PVM in data-buffer
                         to PVM in user-buffer
           when 999     move KELLO in data-buffer
                         to KELLO in user-buffer
     end-evaluate.
     copy PR-VGF2. copy PR-VPB. copy PR-VSF.
  initiate-form section.
   section-start. section-exit. exit.
     copy PR-FUNC.
```

LIITE 2, ESIMERKKI ERÄOHJELMAN TOIMINNASTA

ESIMERKKINÄ ERÄOHJELMA PP7310 - TAPAHTUMALISTA II

Ohjelma PP7310 käynnistetään aloittamalla käynnistysohjelma PP7310A, joka ottaa parametrit näytöltä N7310 ja kirjoittaa niistä jobikortin (tiedoston JP7310) levylle.



Ohjelman PP7310A koodia

```
034500/*********************************************************
034600 C-ERA-AJO SECTION.
034700*********************************************************
034800   OPEN OUTPUT JP7310.
                      ~
035800   WRITE JP7310-RIVI             FROM PARAM-RIVI.
036000   MOVE "!FILE PLISTA=$STDLIST"  TO PARAM-RIVI.
036100   WRITE JP7310-RIVI             FROM PARAM-RIVI.
036300   MOVE "!FILE STDIN=STD7310;TEMP"       TO PARAM-RIVI
036400   WRITE JP7310-RIVI             FROM PARAM-RIVI.
                      ~
051600   MOVE "!RUN PP7310.PROG;LIB=P;STDIN=*STD7310"  TO PARAM-RIVI.
051700   WRITE JP7310-RIVI             FROM PARAM-RIVI.
051900   MOVE "!EOJ"                   TO PARAM-RIVI.
052000   WRITE JP7310-RIVI             FROM PARAM-RIVI.
052200   CLOSE JP7310.
052400   MOVE "JP7310"                 TO JJOB.
052500   IF V-AIKA NOT = SPACES THEN
052600      STRING ";AT=" V-AIKAHH ":" V-AIKAMM
052700                    DELIMITED    BY SIZE INTO AIKAAN.
052800   MOVE "N7310P1"                TO COM-NFNAME.
052900   PERFORM NAYTTO.
```

Kutsutaan erätyön tekevää ohjelmaa ohjelmakirjastosta

```
053000   PERFORM KAYNNISTA-AJOJONO.
```

Ohjelmakirjaston ajojonon käynnistys

```
001000/************************************************************
001100 KAYNNISTA-AJOJONO SECTION.
001200*     ERÄTYÖJONON DOS - KÄYNNISTYS
001300*     tarvitsee palikan COMIM
001400*     käyttöohje:
001500*     MOVE <jobin nimi> TO JJOB.
001600*     PERFORM KAYNNISTA-AJOJONO.
001800*     JJOB         - Eräajo ei muuta
001800*     FERROR       - Eräajo palauttaa virhekoodin
001800*     V-JOBINUMERO - Eräajo palauttaa scriptin nimen
001700************************************************************
```

Kutsutaan komentojonon tekevää ohjelmaa

```
001900      CALL "PPDOSERA" USING
002000                       JJOB
002000                       FERROR
002000                       V-JOBINUMERO.
002500      DISPLAY V-JOBINUMERO,LINE 3,COL 59.
002500      IF FERROR > 0 THEN
002300         MOVE 2          TO V-KOODI
002400         MOVE 1          TO CATSET
002800         MOVE 1          TO CATNO
002900      ELSE
002300         MOVE 2          TO V-KOODI
002400         MOVE 0          TO CATSET
003000         MOVE "Työ käynnistetty" TO CATBUFF.
003300
003100      PERFORM VIRHE.
003100      DISPLAY WINDOW TOP CENTERED TITLE IS "Erätyö"
003100                     BOXED
003100                     POP-UP AREA IS IKKUNAN-TAUSTA.
```

Käynnistetään erätyö

```
003200      CALL "SYSTEM" USING V-JOBINUMERO.
003300      CLOSE WINDOW IKKUNAN-TAUSTA.
```

CALL "SYSTEM" - lause on tapa antaa käyttöjärjestelmäkomento AcuCOBOL-ohjelmasta. Käyttöjärjestelmäkomento on muuttujassa v-jobinumero.

Käynnistysohjelman kirjoittamassa jobikortissa, HP3000.n komentojonossa käyttöjärjestelmäkomennot alkavat huutomerkillä ja Run -komentojen käynnistämien ohjelmien parametrit ovat Run -komentojen jälkeen olevilla riveillä ilman !-merkkejä. Erätyöt lukevat tiedoston riveiltä käynnistysohjelmien antamat parametrit.

Ohjelma PPDOSERA lukee jobikorttia ja kirjoittaa siitä DOSissa komentojonon eli BAT-tiedoston SC7310.BAT tai unixissa scriptin SC7310 + tunniste.

Lisäksi erätyöohjelma tekee input-tiedoston INP7310 ohjelmille POIMI2 ja PP7310, jota nämä ohjelmat lukevat. Parametrit eivät siis Unixissa ja DOSissa ole komentojonossa, vaan erillisessä input-tiedostossa, joka ohjataan erätöille komentojonon ajokomennossa.

PP7310A:n kirjoittama jobikortti, JP7310

```
!JOB JP7310,10007416         .   ;OUTCLASS=LP    , 5,
!FILE LISTA=LP7310;DEV=LP    , 5,
!FILE PLISTA=$STDLIST
!FILE STDIN=STD7310;TEMP
!FILE DHENKILO;TEMP;DISC=77288
!SETJCW RIVILUKUMAARA=22
!RUN PPOIMI2.PROG;LIB=P
VALINTAPARAMETRIT
Yritysnumerot             00-99
                          ~
Laite                     LP
Prioriteetti               5
Sivun pituus              42
!FILE STD7310,OLDTEMP
!RUN PP7310.PROG;LIB=P;STDIN=*STD7310
!EOJ
```

Komentojono, SC7310.BAT ja SC7310.sh

```
runcbl -i INP7310      -o STD7310      -l -e STD7310     PPOIMI2
runcbl -i INP7310      -o STD7310      -l -e STD7310     PP7310
DOS
print \D:LP     LP7310
print \D:LP     PL7310
del dhenkilo
del inp7310
Unix
lp -dlaser -n01 LP73101215
lp -dlaser -n01 P       L73101215
rm dhenkilo
rm inp73101215
```

Input-tiedosto ohjelmille POIMI2 JA PP7310, INP7310

```
VALINTAPARAMETRIT         0046524368
Yritysnumerot             00-99
Osastot                   000000-999999
Henkilönumerot            000000-999999
Henkilöryhmä              1
Henkilöryhmän tarkenne
Henkilöistä mukaan
Kausitapahtumat           E Palkkakausi    00
Vakiotapahtumat           E Pvm:t 010101 - 311299
Rästitapahtumat           K
Päivittäiset tapahtumat   E Pvm:t        -
Yhdistetyt tapahtumat     K
Tulostapahtumat           K Tyyppi
Tietuetunnukset
Käyttäjät
Lajittelu                 1
Laj. organisaatiot
Sivunv. ORG1:ttäin        E
TULOSTUSPARAMETRIT
Ajoaika                           Kopioiden lkm
Laite                     LP      Prioriteetti   5
Sivun pituus              42
```

Ohjelma POIMI2 kerää ajoon henkilöt ja PP7310 tekee tapahtumalistan tiedostoon LP7310 ja tiedoston PL7310 käytetyistä parametreista ja ajon virheistä.

Listaus, LP7310

```
DEMOKONSERNI OY AB  PALKKATAPAHTUMALISTA II  L7310  14:00 SIVU   1
                              TYÖNTEKIJÄT
                        Rästitap., Yhdistetyt tap., Tulostap.
-------------------------------------------------------------------------------------------------------
Pl.nj  Nimi        Y    Tunnit  Yksiköt     A-hinta     Markat KP:t          Tvr  Tappvm Nimi   TaSt
-------------------------------------------------------------------------------------------------------
        KANKAANPÄÄ                      LIISA                          789789     PKP:      4500.00
                                          .
                                          .
***** säänn. tunnit   tunnit yht. yksiköt Ep.al.ansio muut markat vähennykset enn.pidätys maksetaan
```

Ajon parametrit ja virhetiedosto, PL7310

```
TAPAHTUMALISTA II; VERSIO 12/23.10.92
VALINTAPARAMETRIT
0046524368
Yritysnumerot              00-99
Osastot                    000000-999999
Henkilönumerot             000000-999999
Henkilöryhmä               1
Henkilöryhmän tarkenne
Henkilöistä mukaan
Kausitapahtumat            E Palkkakausi    00
Vakiotapahtumat            E Pvm:t 010101 - 311299
Rästitapahtumat            K
Päivittäiset tapahtumat    E Pvm:t        -
Yhdistetyt tapahtumat      K
Tulostapahtumat            K Tyyppi
Tietuetunnukset
Käyttäjät
Lajittelu                  1
Laj. organisaatiot
Sivunv. ORG1:ttäin         E

TULOSTUSPARAMETRIT
Ajoaika                              Kopioiden lkm
Laite                      LP        Prioriteetti                5
Sivun pituus               42


PERFORM M-HTUNNUS-HUOMAUTUS
114100
```